ART DIRECTED FIRE-HAIR SIMULATION

A Thesis

by

CONG WANG

Submitted to the Office of Graduate and Professional Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Chair of Committee,	Frederic Parke
Committee Members,	Philip Galanter
	John Keyser
Head of Department,	Tim McLaughlin

May 2016

Major Subject: Visualization

ABSTRACT

Fire simulation and hair simulation can be used to create stylized characters and character animation in movies. In this research a system was created whereby fire simulation was guided by hair simulation, which this thesis refers to as *Fire-Hair*. This simulation system was built inside Houdini, a professional software package widely used in the visual effects industry. The goal of this research was to develop a workflow that utilized velocity field generated by the hair simulation to drive the fire simulation, and to let simulated fire represent the shape and animation of hair strands. This simulation approach is packaged as a *digital asset* for future use, with all requisite modifiable parameters exposed to artists.

About 20 hair strands were simulated to drive the fire simulation. Hair strand shapes were defined by curves created by the artist; these shapes remain modifiable after creation. Velocity fields which follow hair motion are used as a control field to affect the fire simulation. The final result shows both the physical appearance of fire as well as the shape and motion of hair. The approach was applied to several animated characters to verify reliability and ensure it was visually convincing and robust. The simulated results were rendered using the Houdini built-in render tool, Mantra.

ACKNOWLEDGEMENTS

I would like to thank my committee chair Dr. Frederic Parke for his continuous help throughout the whole process of my thesis. Without your help and patience, this thesis is not possible. I would also like to thank the rest of my committee, Professor Philip Galanter and Professor John Keyser for inspiring me to find new way to expand the possibility of my future work.

Thanks to my friends and family for their support and encouragement. I wouldn't have this chance to do what I love to do without their support.

TABLE OF CONTENTS

Page

ABSTRACT ii

ACKNOWLEDGEMENTS iii

TABLE OF CONTENTS iv

LIST OF FIGURES vi

1. INTRODUCTION 1

1.1 The research problem 1

2. BACKGROUND AND PREVIOUS WORK 2

2.1 Fluid characters in previous movies 2
2.2 Hair modeling and animation 3
2.3 Houdini and operators 4
2.4 Houdini microsolvers 6
2.5 Volume fields in Houdini 7
2.6 Houdini simulation components 7

3. GOALS 14

4. APPROACHES TO CREATING FIRE-HAIR 15

5. METHODOLOGY 18

6. VISUAL ANALYSIS 19

6.1 Hair shape 19
6.2 Hair shading 22

6.3 Hair strand shape 23

7. SIMULATION APPROACH 25

7.1 Hair strand helix 25
7.2 Hair-driven control field 27
7.3 Fire fuel source 33

8. IMPLEMENTATION 36

8.1 Creating hair curve and simulating hair 37
8.2 Control field from hair simulation 45
8.3 Fire simulation implementation 57
8.3.1 Dissipation 61
8.3.2 Shredding 62
8.3.3 Sharpening 62
8.3.4 Disturbance 63
8.4 Render 64
8.5 Results 66

9. CONCLUSION AND FUTURE WORK 68

9.1 Future work 69

REFERENCES 71

LIST OF FIGURES

Page

Figure 2.1 Characters with fluid simulation in movies 2

Figure 2.2 Digital paintings with Fire-Hair 3

Figure 2.3 The node tree for the advect field section in the preset *Pyro Solver* 10

Figure 2.4 The node tree of force update section inside the *Pyro Solver* 11

Figure 2.5 Node tree of microsolvers used by Houdini to simulate combustion 13

Figure 4.1 Steps to get simulated artistic directed Fire-Hair result 16

Figure 6.1 Different hair types 19

Figure 6.2 A comparison between a fire hair painting and two different combustion 20

Figure 6.3 Reference images of campfires 22

Figure 6.4 Different flame colors for different temperatures 23

Figure 6.5 Different hair types with different curliness 24

Figure 6.6 Curliness increases from root to tip 24

Figure 7.1 A basic helix in xyz space 25

Figure 7.2 Different hair curve shape with corresponding profile ramp 26

Figure 7.3 The look of different hair curliness from before simulation 27

Figure 7.4 Influence regions around a hair curve 28

Figure 7.5 Points scattered into the cylindrical space 29

Figure 7.6 Point distribution around a hair curve.....31

Figure 7.7 Steps to convert hair curves to a cylindrical velocity field. 32

Figure 7.8 *vel* field shown as streamers. 33

Figure 7.9 Process to create fire fuel source volume 34

Figure 7.10 Two approaches to get volume from simulated hair curves 35

Figure 8.1 An illustration of the general process used in creating fire-hair renders 37

Figure 8.2 The node network of hair strands control 38

Figure 8.3 The VOP network inside the Helix node. 39

Figure 8.4 Three steps inside the Helix vopsop to get the helix curve.....39

Figure 8.5 Controls exposed to the users for modifying the helix curve's shape 40

Figure 8.6 Instancing hair curves to the head geometry root points. 41

Figure 8.7 How the hair looks after attaching hair curves to the head geometry.....42

Figure 8.8 Hair simulation DOP network 43

Figure 8.9 Essential steps inside the hair simulation DOP network. 44

Figure 8.10 Constraints for attaching hair to head geometry.....45

Figure 8.11 SOP network for obtaining a velocity field from hair curves.....48

Figure 8.12 Flow graph for getting the velocity field. 49

Figure 8.13 Network inside the vopsop. 50

Figure 8.14 Point cloud with velocity attributes obtained from hair curves. 51

Figure 8.15 Velocity field obtained from vopsop 52

Figure 8.16 User control in creating velocity field 53

Figure 8.17 Interpolation drop list.....54

Figure 8.18 Different interpolation types and the point clouds generated 55

Figure 8.19 Change the *vel* field size using the *Sample Distance* parameter.....56

Figure 8.20 Screen shot of geometry used for creating fire source.....57

Figure 8.21 Fire source volume created 58

Figure 8.22 Screen shot of the node tree inside the DOP network 59

Figure 8.23 Processes inside the fire solver DOP network 59

Figure 8.24 Different settings for source volume nodes. 60

Figure 8.25 Setting for shape tab on *Pyro Solver* 61

Figure 8.26 Simulation with different dissipation setting 61

Figure 8.27 Simulation with different shredding settings 62

Figure 8.28 Simulation with different sharpening setting.....63

Figure 8.29 Simulation with different disturbance setting 63

Figure 8.30 Shader setting.....64

Figure 8.31 Final renders. 67

1. INTRODUCTION

1.1 The research problem

We can identify a trend these days in the movie industry where more and more characters are created using fluid simulation. For example, in *Ghost Rider* the fire on the rider's skull was created using fire simulation, in itself a type of fluid simulation.

Fire simulation is one important kind of fluid simulation. In some cases, fire simulation is only used to create the illusion that a character is on fire. In other cases, the very character itself is composed of fire. Characters made up of fire often appear in fantasy movies, in which cases the fire simulation is intended to look as real as possible. However, if the fire simulation can't follow the motion of the source object, it could mask the motion and details of the source object. In such a situation, the audience will not be able to clearly ascertain what is going on under the fire.

In my research, I used simulated hair as the source object and focused on combining the hair and fire simulation, using fire simulation to show the motion of the hair.

Hair is an element that can help create a stylized character and express that character's emotion and uniqueness. There is a need for fire simulation which follows and maintains the motion and shape of hair simulation. Flames should not only move upwards, but follow the shape of the simulated hair with the fire simulation tracking the hair motion. Even when the hair is not rendered we should still be able to see the fire follow the shape and motion of the hair.

2. BACKGROUND AND PREVIOUS WORK

2.1 Fluid characters in previous movies

Many fluid-based characters have appeared in movies. A 1989 fiction-adventure movie, *The Abyss*, gave us a creature made of seawater. This creature could imitate a human face by transforming shape. In *Terminator 2*, the T-1000 had a mimetic poly-alloy (liquid metal) body that allowed it to assume the form of other objects or people of equal volume. In *The Chronicles of Narnia 1* there is a water creature the size of a river. In *Ghost Rider*, the fire on the rider's skull was made using fluid simulation. In *Wrath of the Titans* a human looking giant, Kronos, is created using both fluid simulation and rigid body simulation to make the body look and act like rock with lava flowing on it.

(a) (b)

Figure 2.1: Characters with fluid simulation in movies. (a) The Abyss [1] (b) Ghost Rider [2].

In Figure 2.2, 2d painting artists Sakimichan and Aster-phire use fire texture to replace hair texture; this approach creates very stylized and interesting results.

(a) (b)

Figure 2.2: Digital paintings with Fire-Hair. (a) Fire Vampire by Sakimichan [3] (b) Flame Princess by Aster-phire [4].

2.2 Hair modeling and animation

At least four computational models have been proposed and used for simulating the dynamics of individual hair strands: mass-spring systems [5], projective dynamics [6], rigid multibody serial chains [7, 8] and dynamic super-helices [9].

An early attempt to animate one strand of hair was made by Rosenblum et al [5] using a mass-spring system. Each hair strand is considered as a set of particles connected with hinges as well as stiff springs. Each particle can translate along one direction and rotate about two axes. Hair stiffness is controlled by the angular springs at each joint.

In 1992, Anjyo et al. proposed a model based on one-dimension projective differential equations for simulating the individual hair strands [6]. In the first step, they simulated the characteristics of a cantilever beam to get an initial configuration of every hair strand. After this each hair strand is considered as a chain of rigid sticks. This

method is very efficient. A large number of hair strands can be simulated this way relatively quickly. The implementation is also simple.

In the *rigid multibody serial chains* model, every single hair strand is simplified into a serial rigid open chain with segments known as a multibody [7]. This approach only discusses the twisting and bending degrees of freedom of the multibody chain. Forces accounting for the bending and the torsional rigidity of the hair strand are applied on each link.

More recently, Bertails et al. created a model for the nonlinear behavior of single hair strands with *Dynamic Super Helices [9]*. This method is physically based and it can preserve the length of hair during deformation.

Based on super-helices, Alexandre included frictional contact in an *Inverse Dynamic Hair Model [10]*. In this method, thin elastic rods modeled as super-helices form a collection of guide strands. The linear mass density, stiffness, friction coefficient and elasticity assume typical values for hair. Differences in these properties represent the differences between hair types. The inner forces of the hair during the simulation cause the deformation of hair strands, while external forces, for example the contact friction, help maintain the original shape of the hair. The contact forces, including friction, can be relatively large compared to the inner forces, so hair strands can maintain curliness when they are moving.

2.3 Houdini and operators

Houdini is a node-based software developed by *Side Effects Software [11]* . It is widely used in the movie industry because of its powerful and flexible dynamic

simulation system. Users can build simulation tools and 3D modeling tool within Houdini by connecting different operator nodes. Simulation data is handled and manipulated sequentially based on how users arrange the node connections. The prior research results discussed above informed my use of Houdini operator nodes to simulate both hair and fire.

Compared to other 3D animation packages, for example, Maya, one of the main advantage of Houdini is that it is not a black box system. Users have access to the provided solvers and can modify solver structures to suit their own requirements.

Houdini's operators are divided into the following groups:

- OBJs – Top level nodes that pass transform information. They are usually container nodes that allowed users to build other type operators inside.
- SOPs - Surface Operators. These operators are used for surface generation and procedural modelling.
- POPs - Particle Simulation Operators. These operators are usually used to simulate particles.
- CHOPs - Channel Operators –These are used for procedurally animation. For example, these nodes can be used to change translation and scale values based on noise or audio input.
- COPs - Composite Operators. These operators are used for compositing. User can composite multiply rendered layers using composite operators, and modify color and other values to get an intended result. Users can also use composite

operators to generate noise maps. One usage instance would be to generate a white foam map for procedural ocean surfaces.

- DOPs - Dynamic Operators – These operators are essential for dynamic simulations including cloth and fluid simulations and rigid body interaction.
- SHOPs - Shading Operator – Users can choose dozens of predefined shader as well as build their own.
- ROPs - Render Operators –These are for building networks of different rendering passes (beauty passes, depth passes etc).
- VOPs - VEX Operators –These operators are used for building custom nodes to achieve a desired functionality.

VEX is Houdini's Vertex Expression Language. It is used inside Houdini by expert users for programming. It compiles faster than connecting preset nodes. VEX operators pack VEX code into blocks that artist can connect, just as with predefined nodes.

2.4 Houdini microsolvers

Microsolvers are nodes inside DOPs (Dynamic Operators) which perform specific tasks needed for solving simulations. Each type of microsolver has one specific function. Larger and more complicated solvers can be built up from different microsolvers with different functions. The *Pyro Solver*, discussed later, is one of these larger solvers made up of many microsolvers.

2.5 Volume fields in Houdini

Volume fields can be defined by surface operators (SOPs) or by dynamic operators (DOPs). In Houdini, volume fields come in two types, scalar fields and vector fields. Density is a frequently used scalar field when creating a fire source.

There are three major ways to define custom fields in Houdini. The first requires using a *volume* surface operator. The v*olume* SOP will define an empty volume primitive for further use. The second method uses an *iso-offset* surface operator. This operator will define a volume based on input geometry, which can be a closed mesh or a group of points. The third method is creating a volume field by using a *volumevop* surface operater. *Volumevop* is a VEX operator. This method runs VEX code over a set of volume primitives. The operations are defined by building a VEX operators network based on the user's needs. This method is the most flexible and powerful and by using this method, users can get user defined mathematical functionality. VEX code is compiled and runs much faster than using normal expressions which are interpreted.

2.6 Houdini simulation components

Houdini's *Wire Solver* is useful in simulating long flexible objects – such as wire objects. When wire objects are used with constraints, they can be used to simulate hair. The *Wire Solver* is based on theory that is very similar to Alexandre's research [10]. Wire objects have physical properties such as density, width, friction, elasticity, etc. These properties are input to the *Wire Solver* to compute visually believable simulation results.

The fire simulation in this research is generated using Houdini *Pyro Solver*. Much research has been done in the area of fluid dynamics including fire and smoke simulations. The following detailed explanation helps us to understand the components used in the *Pyro Solver*.

In 3D space, the flow of gaseous fuel and combustion products is modeled using the following Navier-Stokes equations [12]

$$\frac{\partial \vec{u}}{\partial t} + (\vec{u} \cdot \nabla)\vec{u} + \frac{1}{\rho} \nabla p = \vec{g} + \ v\nabla \cdot \nabla \vec{u} \qquad (1)$$

$$\nabla \cdot \vec{u} = 0 \qquad (2)$$

where $\vec{u}$ is velocity, t is time, is pressure, $\vec{g}$ is body forces, v is viscosity

The Equation (1) is called the momentum equation. The equation is a variation on Newton's second law of motion $\vec{F} = m\vec{a}$ [13], it tells us how the fluid accelerates based on the forces acting on it. Equation (2) is the incompressibility condition and tells us that the divergence of the velocity must always be equal to zero. This prerequisite makes sure that the sum of all the velocities in all directions entering and leaving at a point in space is zero. The total mass of the fluid is conserved when velocity is conserved over the entire simulation [14]

Three main steps are needed to solve the Navier Stokes Equation.

1. Advection
2. Diffusion and external forces

3. Pressure/ Incompressibility

Advection which represented as $(\vec{u} \cdot \nabla)$ in the equation above means that the motion of the fluid drives the motion of the entities inside it. When one specific part of the fluid can move the adjacent part of the fluid along, we call this self-advection. Houdini can make use of *Gas Advect* microsolver to advect one or more fields at the same time. This can be seen in the node tree for the advect field section inside the preset *Pyro Solver* (a solver created directly by using the predefined Houdini tool without any modification and customization) in Figure 2.3.

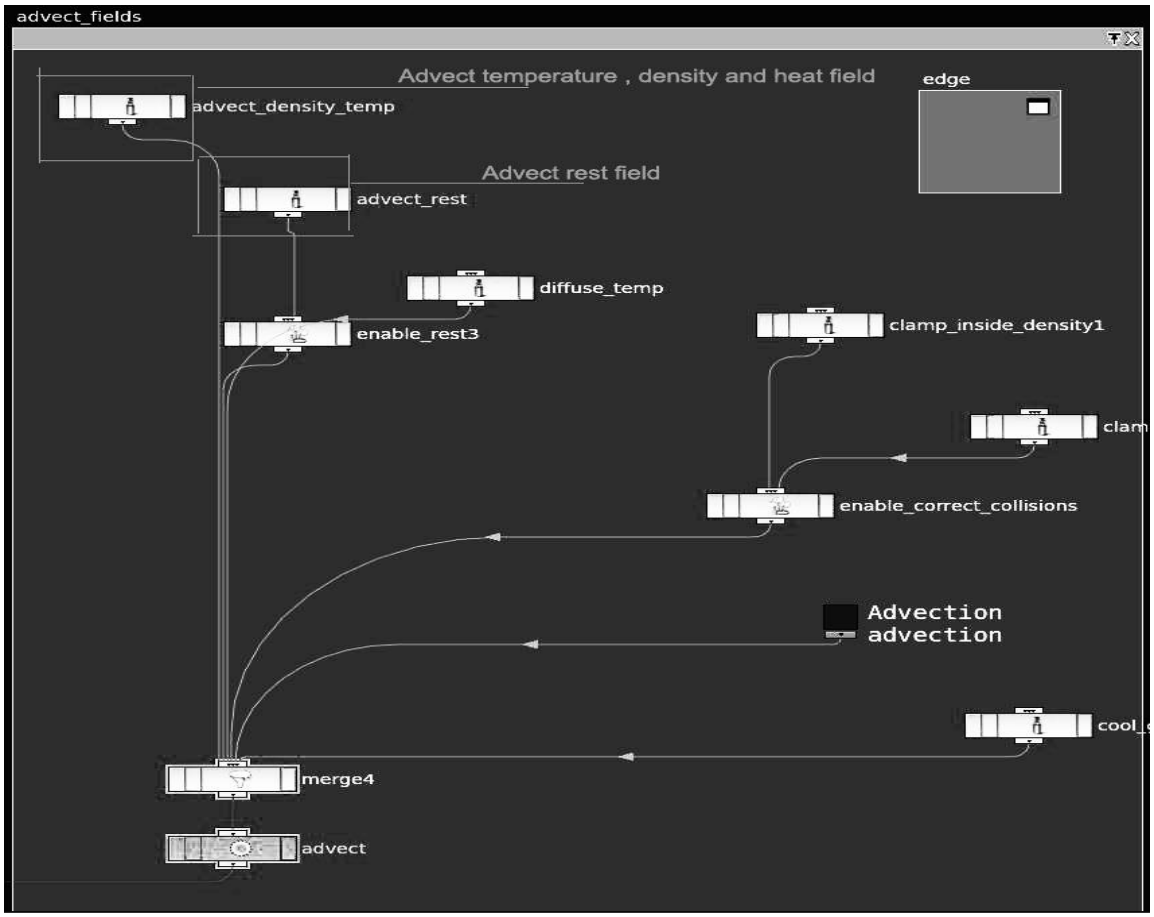


Figure 2.3: The node tree for the advect field section in the preset *Pyro Solver*. The nodes inside red boxes are Gas advect microsolvers to advect temperature, density and heat fields.

Diffusion is the $v\nabla \times \nabla u$ term in the equation (1). It allows velocity to propagate outwards from the current location. The speed of this is controlled by the viscosity parameter. Viscosity is defined as the measure of a fluids resistance to deformation and flow[15]. A high viscosity value will result in slower and thicker fluids, while a low value results in more dynamic and lively fluids. Inside Houdini, the *Gas blur* microsolver defines and controls the viscosity of the fluid.

The $\vec{g}$ term is the sum of all external forces[15]. Wind, drag force and gravity are some typical forces, but this term may also include forces that require separate simulation. The user can add control of the external forces by using defined volumetric fields. We use a number of Gas Microsolvers to represent these forces, merging them all together as the total force. Some typical microsolvers that are part of the node tree in the forces section in the preset Pyro Solver are shown in Figure 2.4.

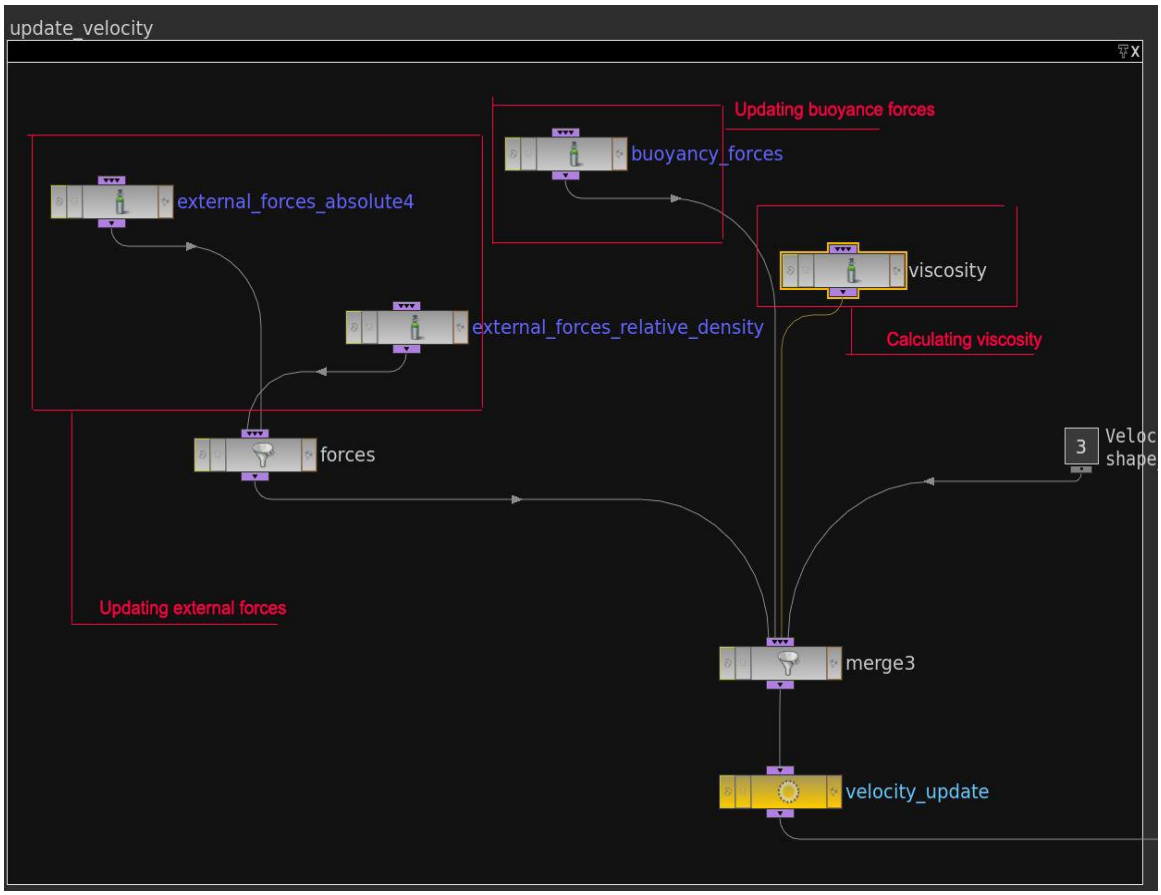


Figure 2.4: The node tree of force update section inside the *Pyro Solver*.

The pressure p defined as the force per unit area[15]. Higher pressure areas will exert force on lower pressure regions by having outward motion.

$\nabla \cdot u = 0$ is the incompressibility condition.

During the simulation process, the fluid will lose some velocity, due to weighted averaging steps which occur during advection. This results in loss of some details like turbulence seen in natural smoke and fire. In Houdini, there are microsolvers that can be used together to artificially restore this lost data. This process measures the vorticity and adds artificial rotational forces. The added force will spin the flow field in a particular direction, giving the user control of vorticity. We need to point out that the force added here is not the actual lost data but is a user defined value to approximate the lost details.

In Houdini's *Pyro Solver* fuel, temperature and velocity are each stored as volume fields. Fuel has an associated ignition temperature *Ti* and a burn rate *b*. We use *voxel (i, j, k)* to represent a unit volume, at a given time step. If *voxel (i, j, k)* has fuel concentration $f(i, j, k) > 0$ and temperature $T(i, j, k) > Ti$ then a quantity of fuel $\Delta f(i, j, k) = \Delta t b$ is consumed and removed from *voxel (i, j, k)*. Here Δt refers to the simulation time step. Temperature $T(i, j, k)$, divergence $\varphi (i, j, k)$ mentioned above and *density* $\rho(i, j, k)$ are modified according to how much fuel is consumed. To make the flames more dynamic and realistic, synthetic turbulence can be added into the velocity field [16, 17]. A additional velocity field can be used to drive the motion of the combustion [18]. I am using a velocity field based on the simulated hair motion.

These steps are calculated and handled by several related microsolvers inside Houdini; the nodes tree is shown in Figure 2.6.

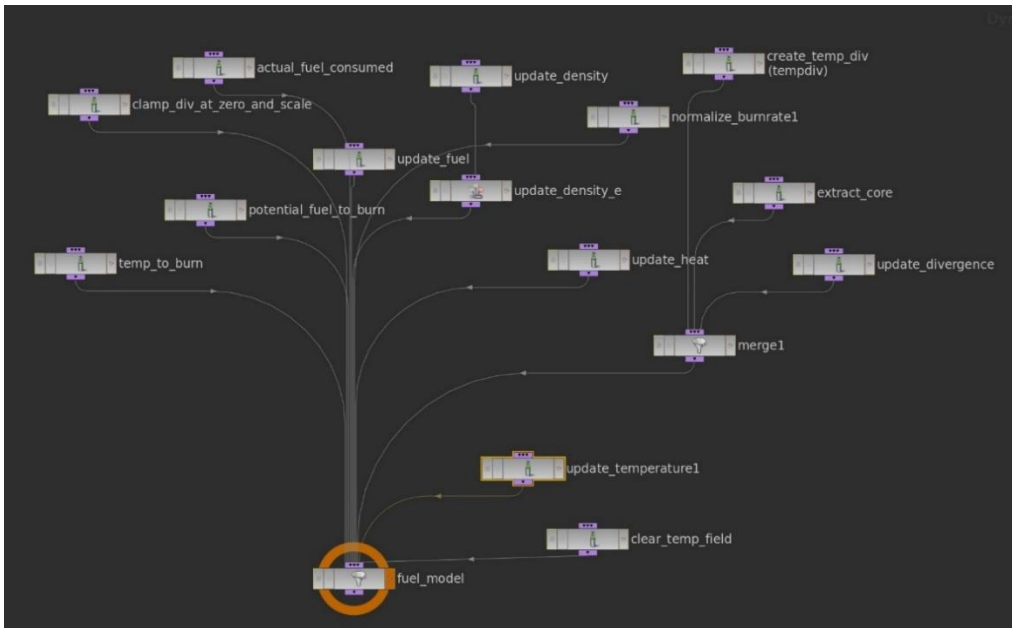


Figure 2.5: Node tree of microsolvers used by Houdini to simulate combustion.

3. GOALS

The goal of this research was to create a hair simulation system and use this system to drive fire simulation. Using this system, we can create stylized characters that have a moving and artist-controlled fire element resembling both hair and fire.

This research had the following objectives.

1. Create convincing hair simulation.

2. Create fire simulation which followed the shape and motion of the hair. For example, if the hair moves downwards, the flames should follow it.

3. Create the real physical look of fire.

4. Provide artistic control.

4. APPROACHES TO CREATING FIRE-HAIR

There are two candidate approaches which achieve similar visual results for Fire-Hair as shown in the digital paintings in Figure 2.2. One is to simulate the hair curves and base the fire simulation on it. The other is to simulate the hair curves and convert them into volumes with animated noise to approximate the physical appearance of fire. For the second approach we just need to simulate the hair and this simulation runs fairly fast. There is no need to simulate the fire.

Approach type	Step 1	Step2	Advantage	Disadvantage
1. Fire simulation approach	Simulate 20-40 hair curves	Simulate fire driven by control field from hair geometry	Easier to get more realistic details	Simulation takes a long time; simulated result takes massive storage space
2. Non-Fire simulation approach	Simulate 20-40 hair curves	Convert simulated hair curves into geometry with animated deformation and render as volume	No simulation time needed, result is more predictable	Could be challenging to get realistic details

Table 4.1: A comparison between two fire approaches

In the first approach, a highly detailed fire simulation could provide realistic shapes and motion which are hard to get from procedural noise or other approaches. The

drawback is this method requires massive storage space and the simulation can take a long time. Also, the simulation is harder to control and shape to the artists' direction when compared to the non-fire simulation approach.

The second approach could save a lot of simulation time through converting hair curves into volumes with procedurally animated noise, and would be more predictable for the artists. But this approach can be very challenging in realizing the realistic look and dynamics of fire.

To choose the better approach, we use campfires pictures in Figure 6.3 as reference. The references show fine details of fire. To get a similar result in our project, we picked the first approach which used fire simulation as we felt it better met our need.

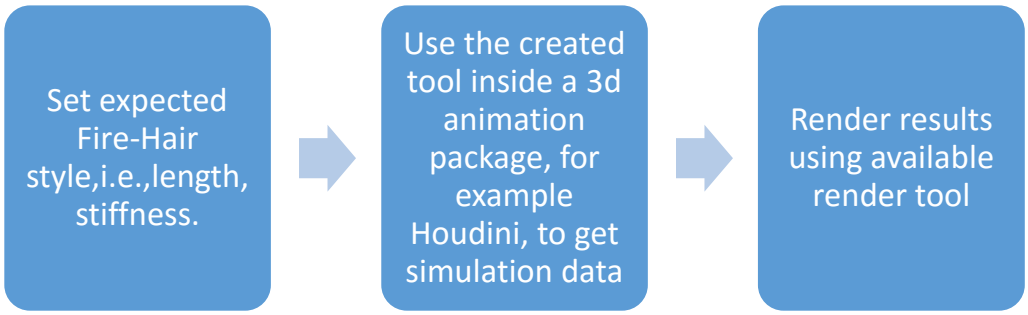


Figure 4.1: Steps to get simulated artistic directed Fire-Hair result.

As shown in Figure 4.1, to get the desired result, we built and used a tool inside Houdini. The term *digital asset* is a user created Houdini tool which can be easily

configured to meet our needs. The digital asset we created receives user inputs, defines and simulates the hair curves, and delivers the simulated hair curves to the fire simulation module to get the final fire simulation. The render settings are also changeable to give users additional control.

5. METHODOLOGY

This research has been done by developing a prototype implementation of a fire-hair simulation approach. This prototype has been used to create several example simulations to verify that the approach facilitates effective, artist controllable fire-hair visual results.

6. VISUAL ANALYSIS

Apart from research discussed in previous sections, visual reference art work and photographs were collected to study the visual elements of fire and hair motion.

6.1 Hair shape

Hair shape can be different based on stiffness, length, density and character motion. Figure 6.1 shows several different types of hair.

(a) (b)

(c) (d)

Figure 6.1: Different hair types. (a) and (b) show smooth hair with clearer strands, (c) and (d) show dry and frizzy hair with less clear strands.

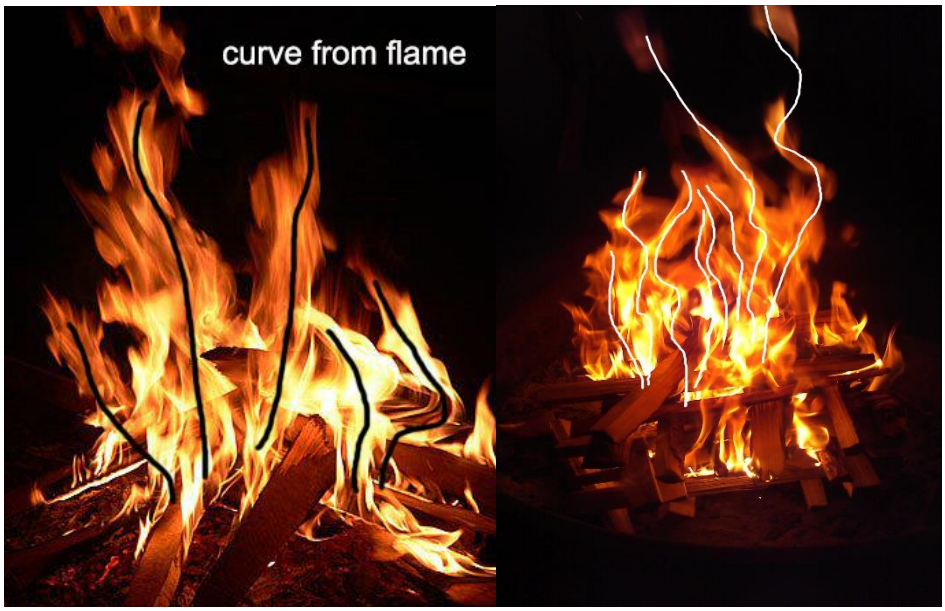
As shown in Figure 6.1, the smooth hair is more dynamic and defines the shape of the hair strands better. We felt that 6.1(a) and 6.1(b) were visually closer to the look of fire.

Figure 6.2: A comparison between a fire hair painting and two different combustion.

One important feature of fire is its dynamic motion. The motion and shape of flames help distinguish fire from other fluids like dry ice vapor or smoke. In this project, we wish the final result to make sense to the audience and make them believe the fire is representing hair. We need to show dynamic fire simulation motion similar to the reference hair images. Since smoother hair shows this dynamic motion better, we used Figure 6.1(a)and (b) as our hair reference.

By observing Figure 6.1(a)and (b), the roots of each hair strand are more convergent and the tips are scattered into small strands. In Figure 6.2, the digital painting of the Fire-Hair shows this same feature.

By comparing different types and scales of fire, we chose a campfire rather than larger-scale combustion as reference because large-scale combustion doesn't possess the curve shaped flames needed to represent hair strands. In Figure 6.3, the base part of the fire which is closer to the fire source has a bolder shape, while the tips of the flames have more flares. With the help of motion blur, each flame actually forms a curve. We can use this visual feature to represent the hair strands in our fire-hair system. This similarity between a campfire and hair makes the campfire images a good reference for us to determine the scale and other parameters of the fire simulation.



(a) (b)

Figure 6.3: Reference images of campfires.

6.2 Hair shading

Based on visual observation of both digital paintings and campfire reference images, the color of fire is basically a ramp from bright yellow to deep brown red. Smoke should be gray to almost black, according to the lighting conditions. Flame color is temperature related in reality. The Figure 6.4 shows a fire color palette with different colors for different temperatures.

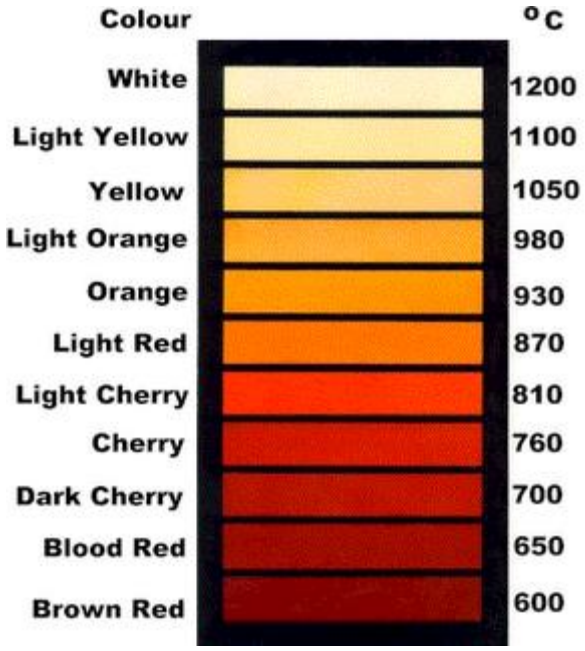


Figure 6.4: Different flame colors for different temperatures.

Since the goal of our fire-hair system is to be artist directed, we want to give the user the option of using physics-based shading or artistic directed shading. The difference between these shading methods is: physics-based shading sets the color based on temperature data from the simulation, while artistic directed shading will render fire color based on a color ramp given by the artist. The second option is more flexible and users can easily adjust the color as needed, or even use totally different colors if desired.

6.3 Hair strand shape

Different people have different hair types, some straight and some curly and everything in between. Figure 6.5 shows hair with different curliness.

Figure 6.5: Different hair types with different curliness[19].

We want our fire hair system to support different hair types with different degrees of curliness. We need to get the desired curliness at the hair simulation stage. By comparing digital paintings and real life hair curliness, we find that in most cases the root of the hair strand has less curliness, and the tips have more.

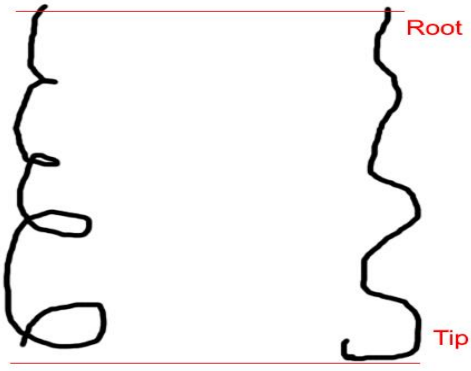


Figure 6.6: Curliness increases from root to tip.

7. SIMULATION APPROACH

7.1 Hair strand helix

We use a basic helix with radius profile ramp to control the shape of the hair curve. The profile ramp is a user-defined curve allowing users to draw controlling points. Figure 7.1 shows a basic helix shape.

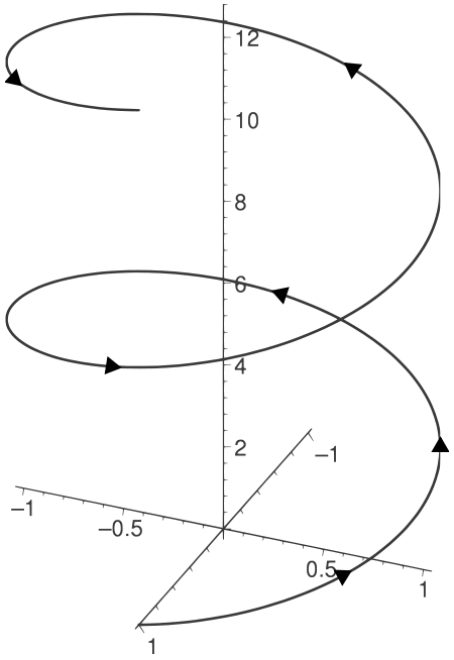


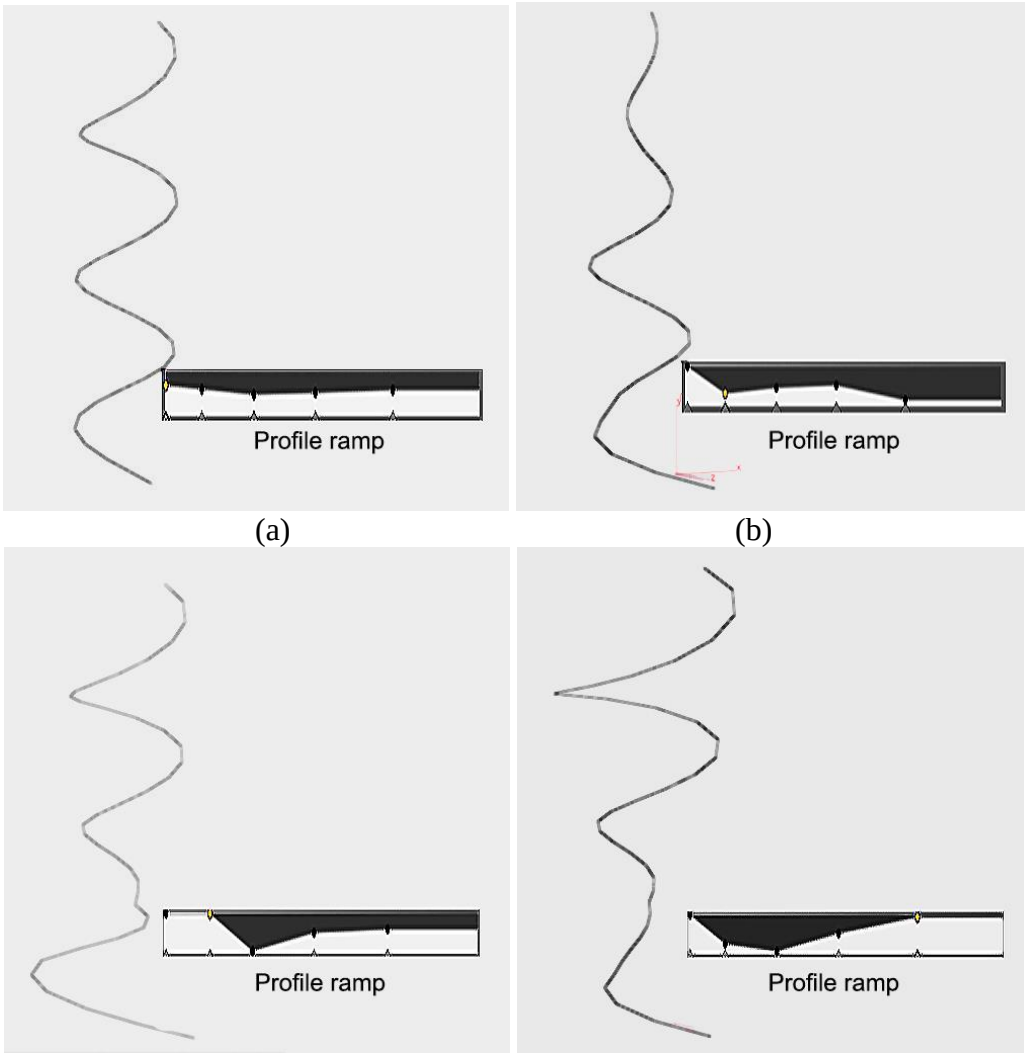
Figure 7.1: A basic helix in xyz space.

The z axis is the up direction. The value i_z below is the radius value from the strand profile ramp given by the user, it can vary with z. The values of x and y are:

$$\mathrm{x} = \cos(z) * i_z$$

$$y = \sin(z) * i_z$$

Different profile ramps return different helixes. The Figure 7.2 shows some different helixes we can get by using different profile ramps.

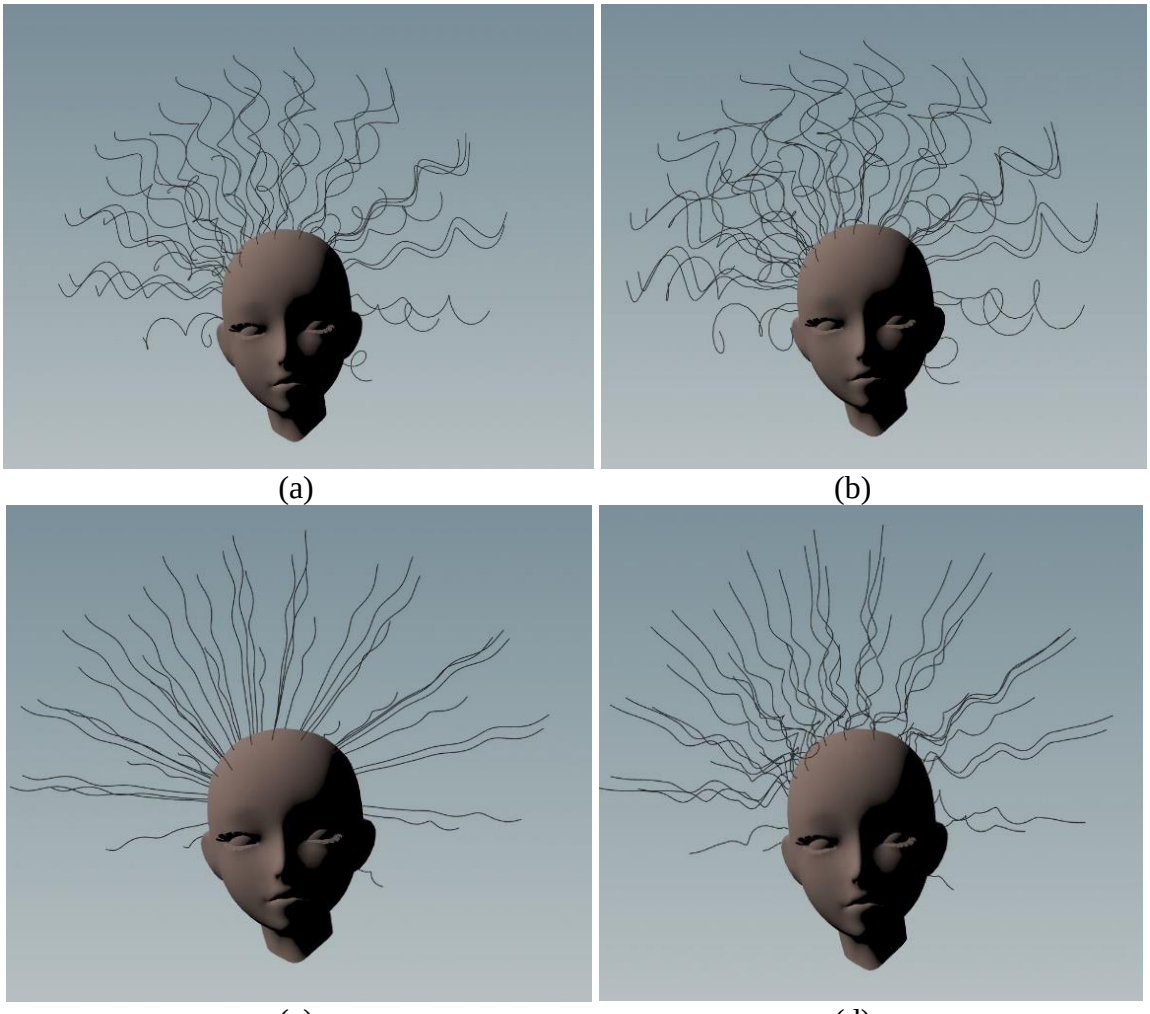


(a) (b)

(c) (d)

Figure 7.2: Different hair curve shape with corresponding profile ramp.

Figure 7.3 shows the results we get after applying the strand helices onto a head. We will explain how to implement this in the *Implementation* chapter.



(a) (b)

(c) (d)

Figure 7.3: The look of different hair curliness before simulation.

7.2 Hair-driven control field

To drive the fire simulation and make the fire follow the curves of the hair strands, we need to create a control field around the simulated hair curves.

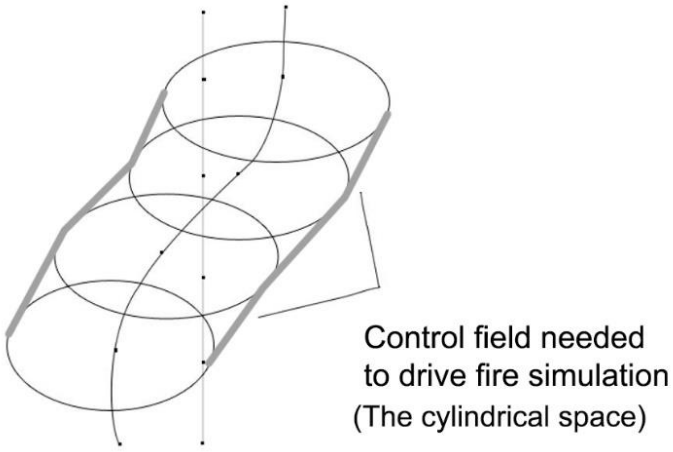


Figure 7.4: Influence regions around a hair curve.

As shown in Figure 7.4, the control field is a cylindrical space which surrounds each hair curve, and follows the hair strand deformation. The radius of the control field is user-defined, enabling the user to decide how closely the fire simulation should follow the hair curve while at the same time avoiding potential control field overlaps.

To get this control field, we distribute many points around the hair curve, and use this point cloud as the basis for our control field.

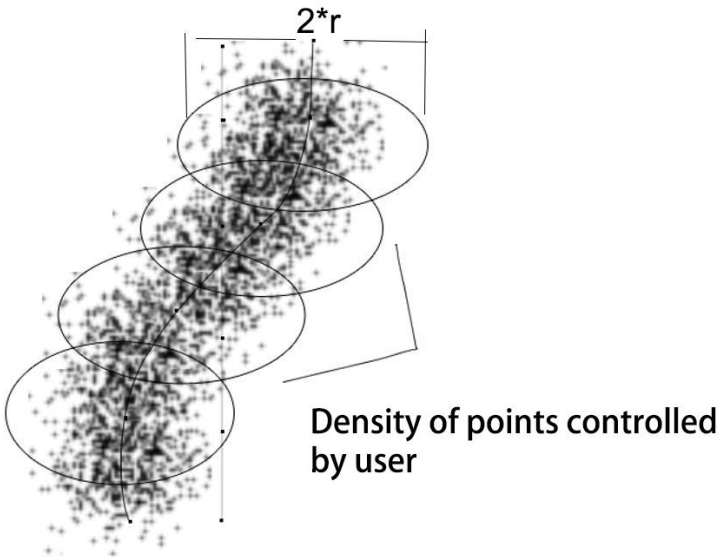


Figure 7.5: Points scattered into the cylindrical space. The radius *r* of field is not a constant. It can be changed based on a profile specified by the user.

Along each hair curve, the cylindrical space may have different radii at different positions. This can be a controlled by user-defined profile as shown in Figure 7.6.

To create the point field, we resample the hair curve to obtain 1000 points which we will then scatter. If necessary, the user is able to set a higher number of resampling points. We then move the resampled curve points to new positions based on a random function and a user-defined ramp. This process will result in the resampled points scattering into the space surrounding the hair curve.

$$X_{new} = (X + displace_x(random(point\ number))) * control\_field\_profile * control\_field\_size$$

$$Y_{new} = (Y + displace_y(random(point\ number))) * control\_field\_profile * control\_field\_size$$

$$Z_{new} = (Z + displace_z(random(point\ number))) * control\_field\_profile * control\_field\_size$$

In these equations, $displace$ is a Houdini displacement function to get the displacement values along the x,y,z directions. *random* is Houdini's building randomize function based on its parameter values. The point number is used as the random function's input value, with each point possessing a unique point number. $control\_field\_profile$ is a ramp defined by the user to control the shape of the scattered space. $control\_field\_size$ is a user controlled multiplier to control the overall scale.

In Houdini the velocity of each point is stored as a velocity attribute. We can set this attribute value to any value as we need. When we sampled the simulated hair curve, we used curve tangent vectors to set the velocity attribute *V*. After moving each point to its new position, we multiply its velocity with a user-defined velocity scale and add a curl-noise[20] based on the point's new position. Curl-noise is a type of turbulence noise based on Perlin noise[21]. This type of noise is often used to procedurally animate turbulent fluid. After this we multiply the result with a user-controlled velocity profile ramp so the user can vary the velocity along the hair curve. We define the new point velocity as V_{new}:

$$V_{new} = (V * Velocity\_Scale + Curl\text{-}Noise_{new\_position}) * velocity\_ramp$$

Now we have 1000 points around each hair strand in their new positions and with new velocity values. This process is illustrated in Figure 7.8.

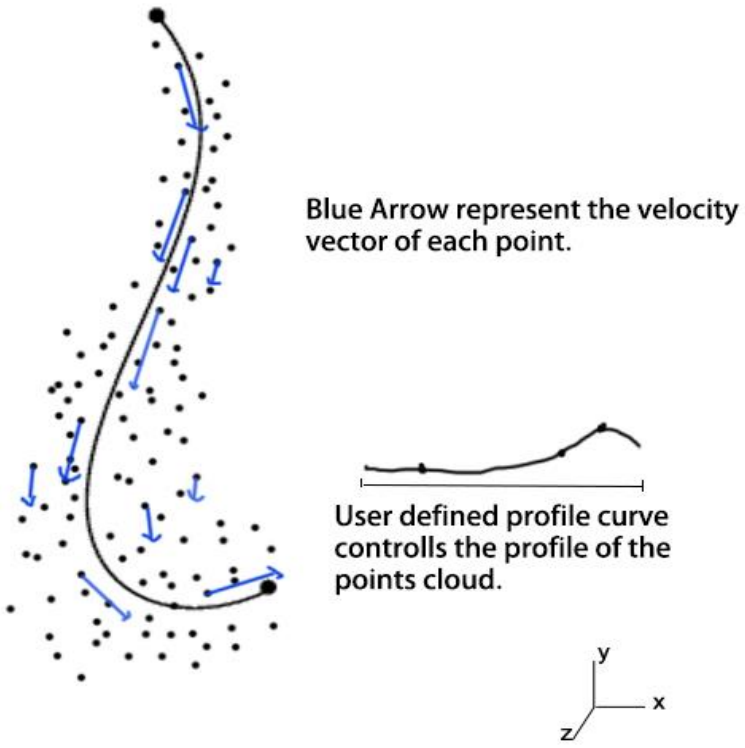


Figure 7.6: Point distribution around a hair curve. The shape is based on the radius profile ramp.

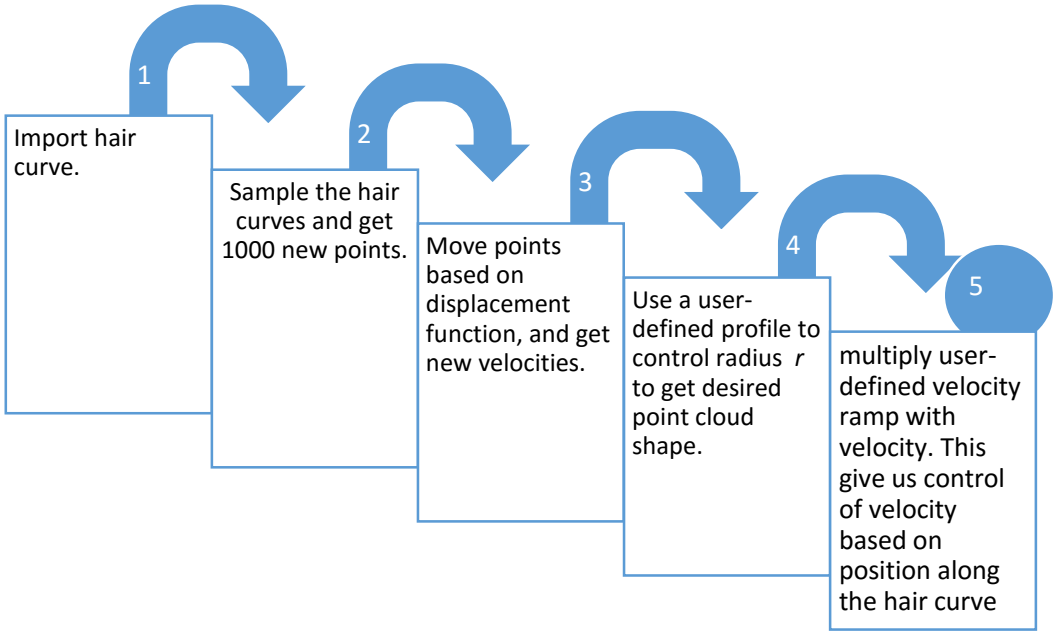


Figure 7.7: Steps to convert hair curves to a cylindrical velocity field.

After we get scattered points, each with a *velocity* attribute, we use a Houdini tool to create a velocity vector field based on these points. By adding this velocity field into the velocity data of each simulation time step, we can use the hair velocity field to affect the fire simulation and make fire motion follow the hair curves.

A three dimensional fluid container, which is a large 3D voxel cells array, was created large enough to contain the hair points. Each cell of the field was given a vector attribute called *vel*, representing the average velocity of all the points included in the cell. This *vel* value is added to the fire simulation velocity so the fire will have a velocity along the hair curves. The ratio of the velocity from fire simulation with that provided by the hair *vel* field is 1:1. This is a user controllable ratio. The Figure 7.8 shows the view of the *vel* field as streamers.

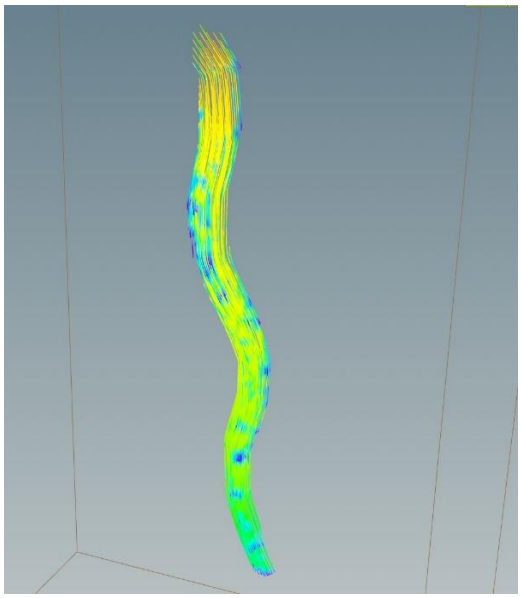


Figure 7.8: *vel* field shown as streamers.

7.3 Fire fuel source

We created the fuel source based in the simulated hair curves. The basic process involved first using a built-in Houdini method to convert each hair curve into a geometric area. Then we converted the area into a volume and used this volume as the source of fuel for our fire simulation.

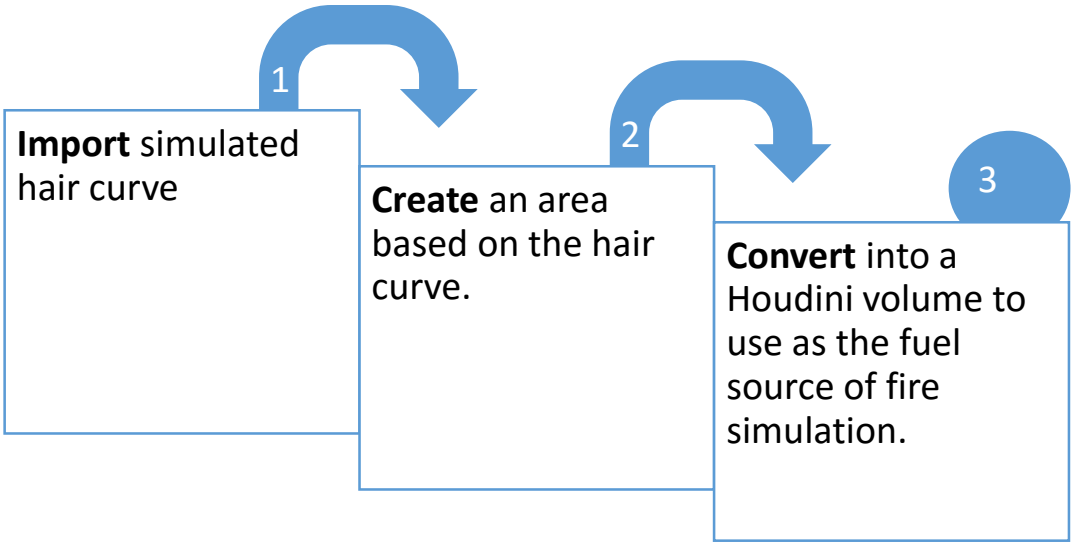
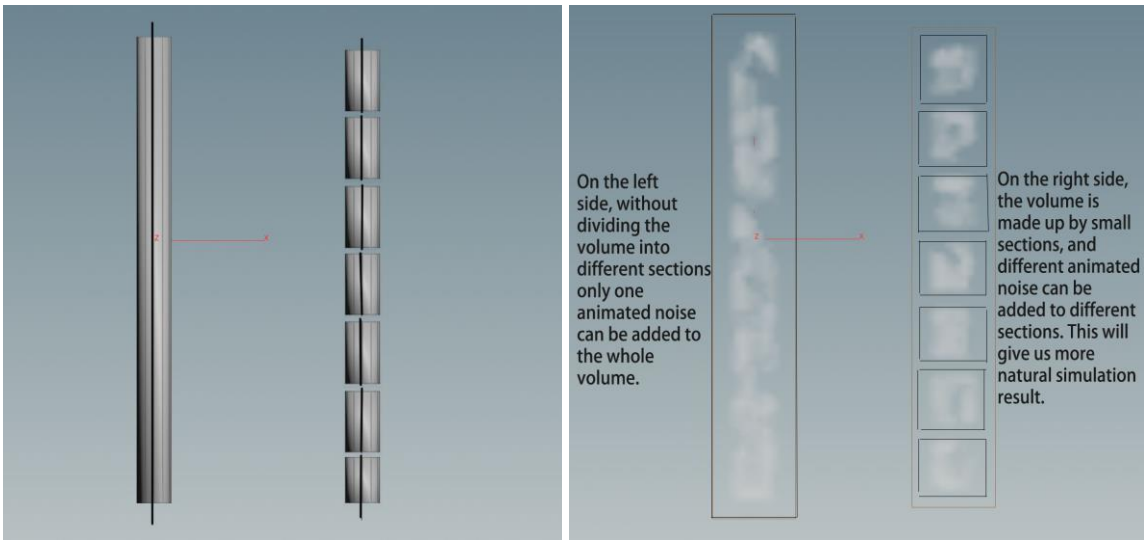


Figure 7.9: Process to create fire fuel source volume.

We tried alternative approaches for creating fuel volume. The first was to make each hair strand into a single cylindrical volume to be used as the source. Another approach was to divide the hair into several segments. Then each segment would be separately converted into a volume. We needed to give the fire source animated noise to reduce its uniformity. Here is a comparison between these two approaches.



(a) (b)

Figure 7.10: Two approaches to get volume from simulated hair curves. (a) shows the cylinders we get from the two different approaches. (b) show the Houdini volumes with animated noise we created based on the cylinders in image (a).

As shown in Figure 7.10(a), on the left is the first approach using a curve as a single geometry and creating a cylinder from it. On the right is the result of resampling it, dividing the curve into several segments and creating cylinders from each separately. The benefit of the second approach is that we see less uniform noise. The first approach showed animated noise pattern moving in a single direction. Also, with moment this volume will present a clear and uniform boundary. These drawbacks result in less interesting fire simulation. In the second approach we can apply noise to each segment separately and can give the noise different direction, resulting in a simulation that is more interesting and more aesthetically appealing.

8. IMPLEMENTATION

The simulation environment used was *Houdini FX 14.0*. Simulation results were rendered using the Houdini built-in renderer *Mantra*. The whole system has been packed into a single *Houdini Digital Asset*. Future users will be able to conveniently create fire-hair effects and make customizations by simply changing several parameters.

The hair strands were simulated using the Houdini *Wire Solver*. Each simulated hair strand produced data. Points on each hair curve get position attributes and velocity attributes. These attributes were then converted into a control velocity field used to drive the fire simulation. Fluid buoyance and drag force, were also used to achieve a more realistic fire look.

The fire-hair system was applied to several animated characters, each with different hairstyles. We tested the robustness of the system by using different animations to ensure the fire-hair system was able to handle different hairstyles and dynamic motions.

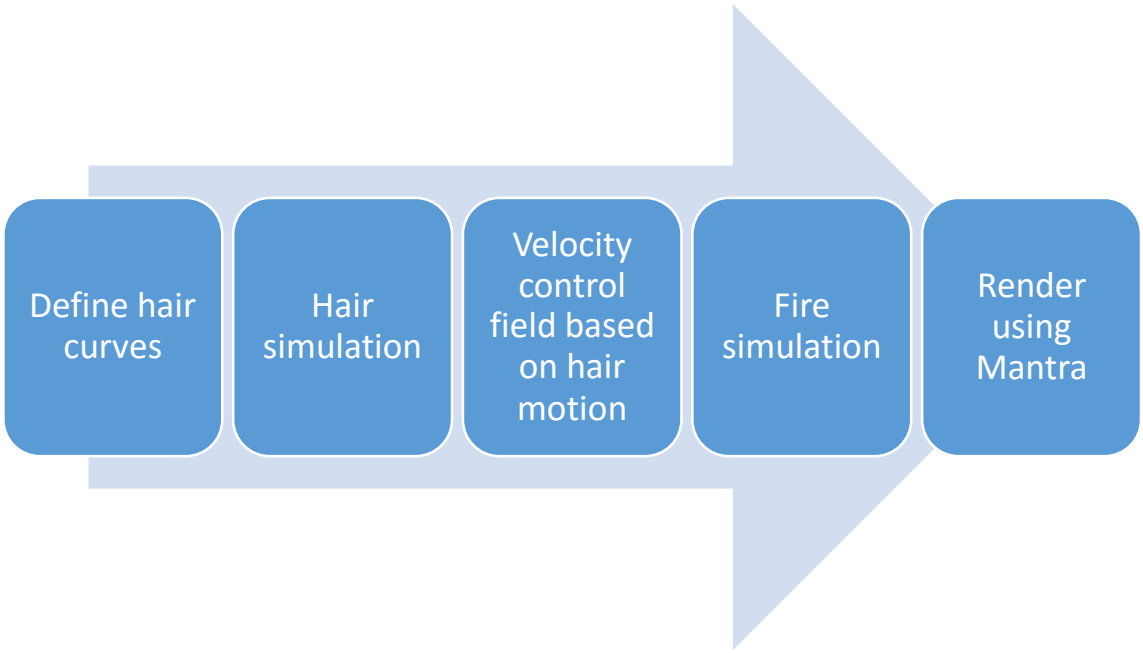


Figure 8.1: An illustration of the general process used in creating fire-hair renders.

8.1 Creating hair curve and simulating hair

This system allows users to create and shape hair curves procedurally, with parameters exposed to artists for easy control. Artists can create the hair curve either from a basic helix shape or by drawing a curve and using it for the basic shape of hair strands. The Houdini network for this functionality is shown as Figure 8.2.

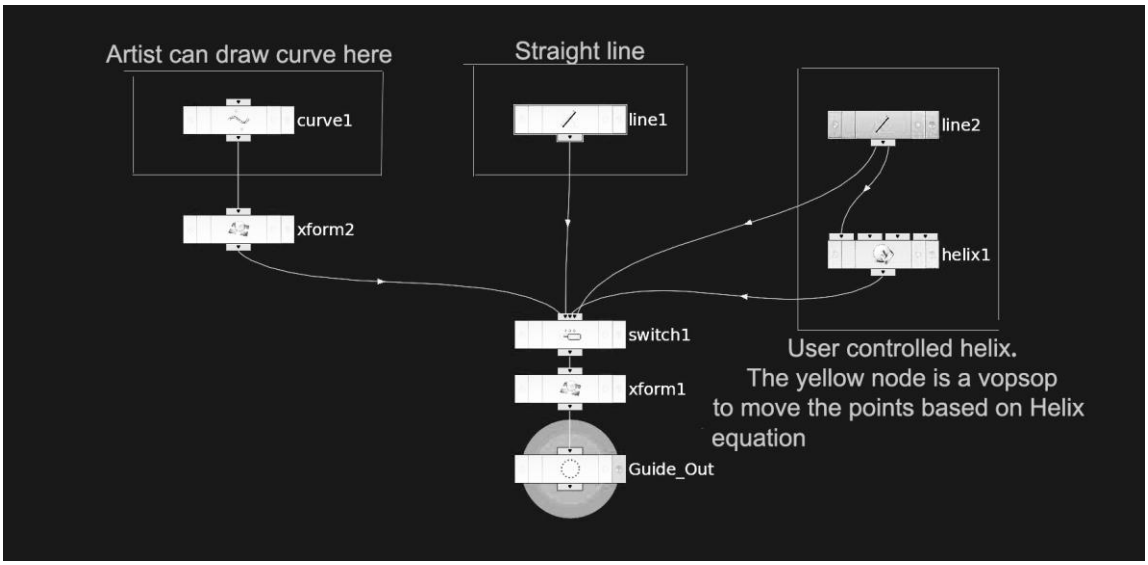


Figure 8.2: The node network of hair strands control.

The default hair strands are defined by a vopsop named Helix. Vopsop is a surface operator that contains a VOP network and can be used to manipulate the attributes of the hair strand points. Each node has a specific mathematical function. Together these nodes perform complex calculations. The math behind the network has been discussed in the previous chapter. The artist can control the number of divisions, number of turns, radius, length and the profile of each hair curve. These parameters are exposed to the users. The Figure 8.3 shows this VOP network. Figure 8.4 is a flow graph of the algorithm inside the VOP network.

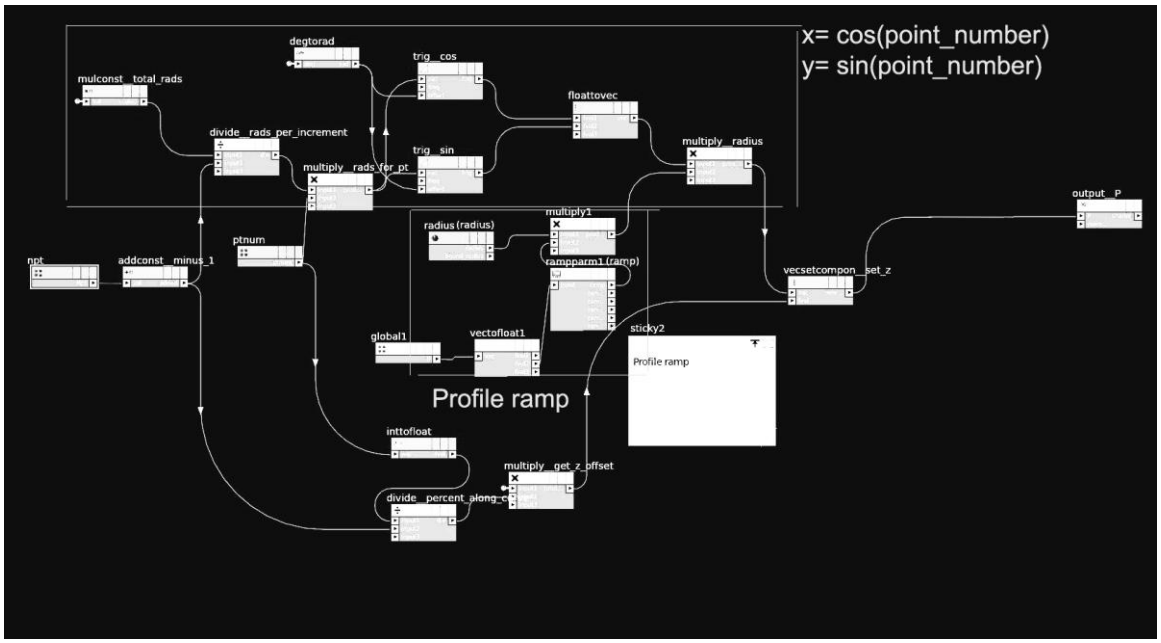


Figure 8.3: The VOP network inside the Helix node. The three red boxes represent three different modules with specific functions.

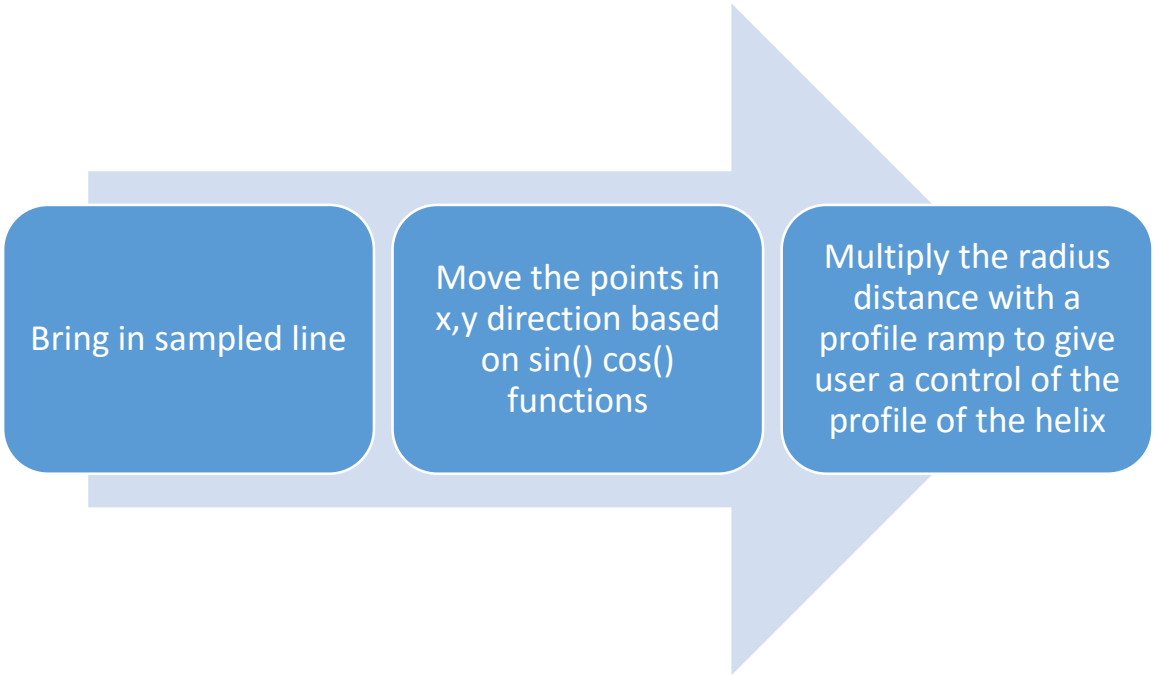


Figure 8.4: Three steps inside the Helix vopsop to get the helix curve.

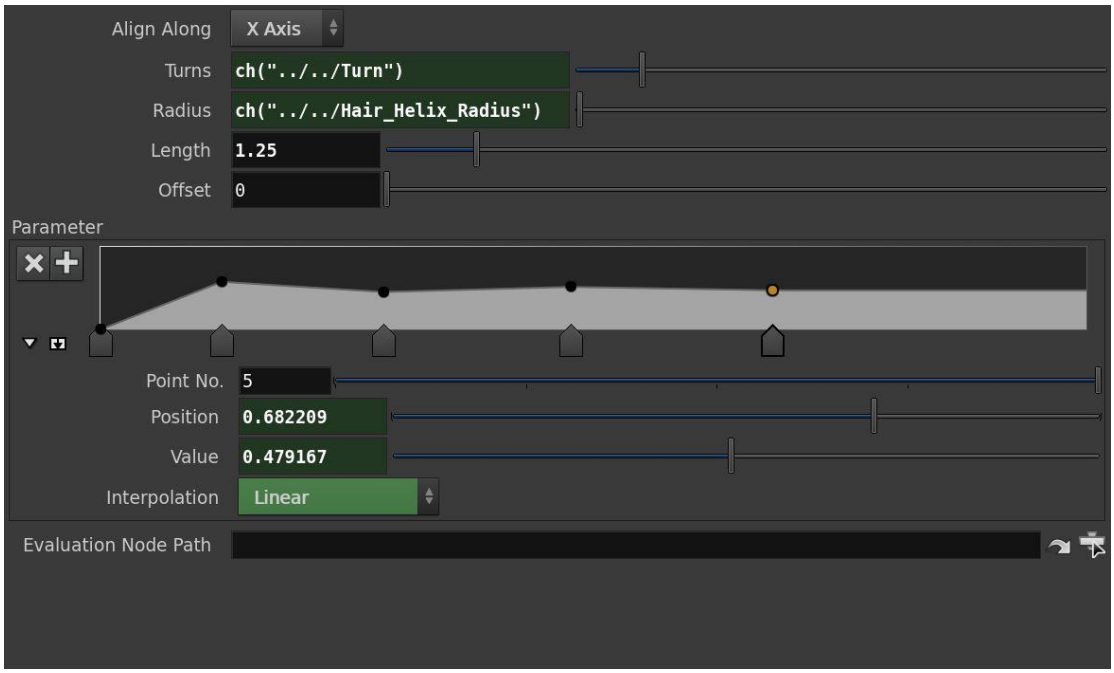


Figure 8.5: Controls exposed to the users for modifying the helix curve's shape.

We exposed several controls to the users so the helix curve shape could be easily changed. As shown in Figure 8.5, the parameter *Turns* controls how many turns the helix will have. *Radius* controls the overall radius of the helix. *Length* control the distance from the head point to the end point. The *profile ramp* is used to control the profile of the helix.

We need a number of hair curves for the simulation. Our system allows the user to create customized curves, but in this thesis we used the helix for the examples. The head geometry has points where hair curves attach. We copy the single hair curve we designed to each of these points. We used about 40 hair curves on each animated character. The number of hair curves used depends on the desired results. A small number will save simulation time but will leave empty space between hair strands. This

will make the fire hair very sparse and will not represent hair motion very well. A large number will significantly increase the simulation time, and if they overlap too much, hair strands in the final simulation will not be distinguishable enough.

The points where the hair curves connect to the head geometry are the *root* points. We group these root points altogether. The Houdini simulation network can recognize and use the points inside this group to constraint hair curves to the head geometry.

The Figure 8.6 is the Houdini nodes network for instancing the hair curves to the root points.

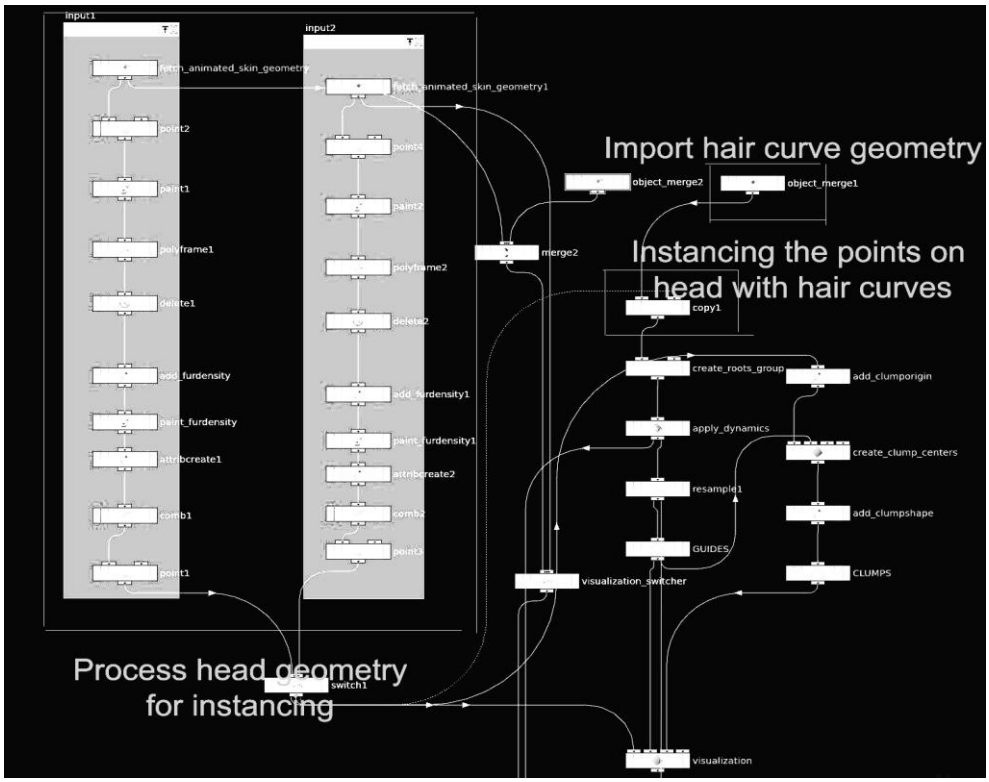


Figure 8.6: Instancing hair curves to the head geometry root points.

Figure 8.7 shows the result after applying the hair curves to head surface.

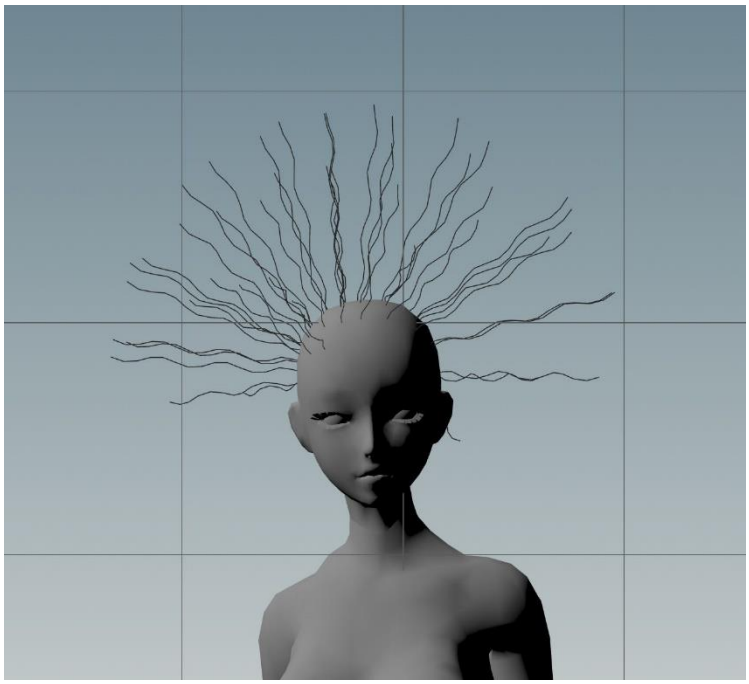


Figure 8.7: How the hair looks after attaching hair curves to the head geometry.

The head geometry and hair geometry are now ready to be sent into the DOP (Dynamic operator) network for simulation.

As shown in Figure 8.8, the hair curves are imported into the DOP network by using a *wire object* node. The head geometry is imported as a *static object* so it will only be used as a collider and won't be affected by the simulation. The *roots* group is attached to the head surface using *wireglueconstraint*, a constraint to fix both rotation and translation of the root points to the head geometry.

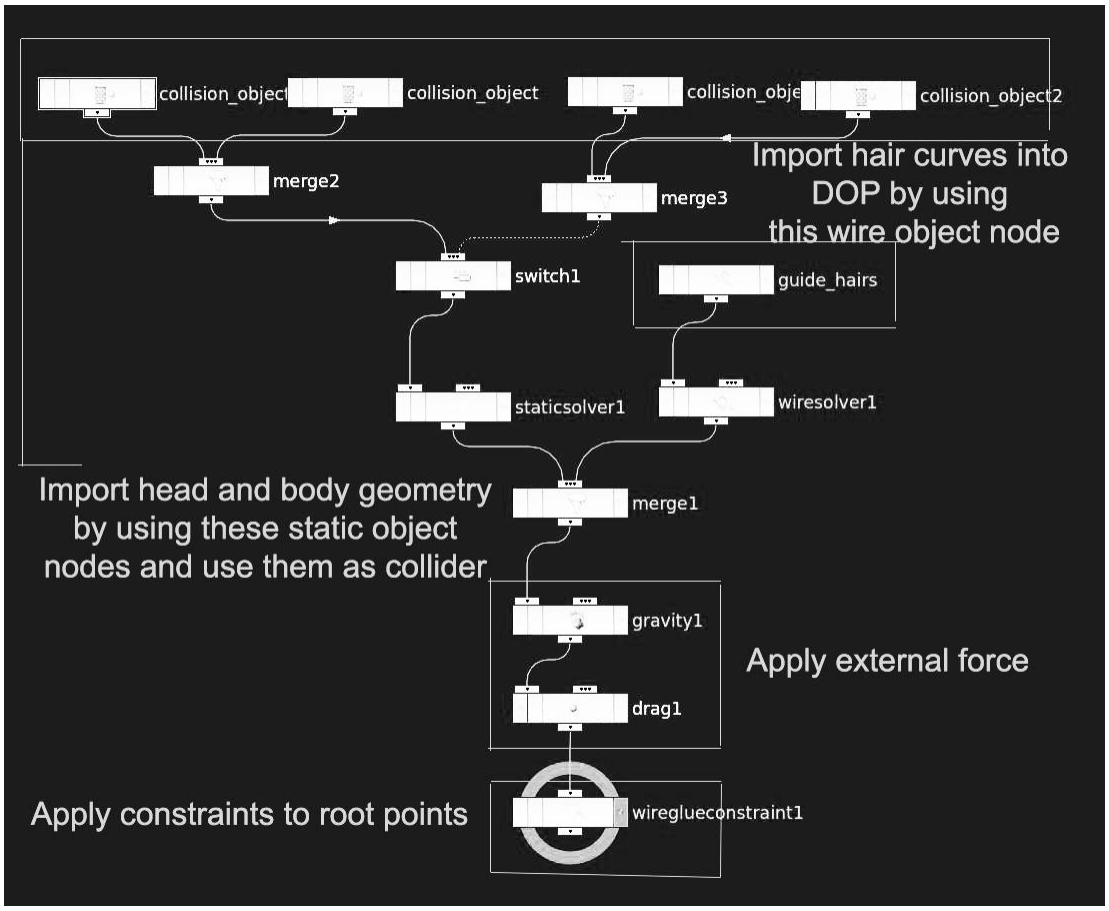


Figure 8.8: Hair simulation DOP network.

This simulation process can be abstracted into a general flow diagram as shown in Figure 8.9

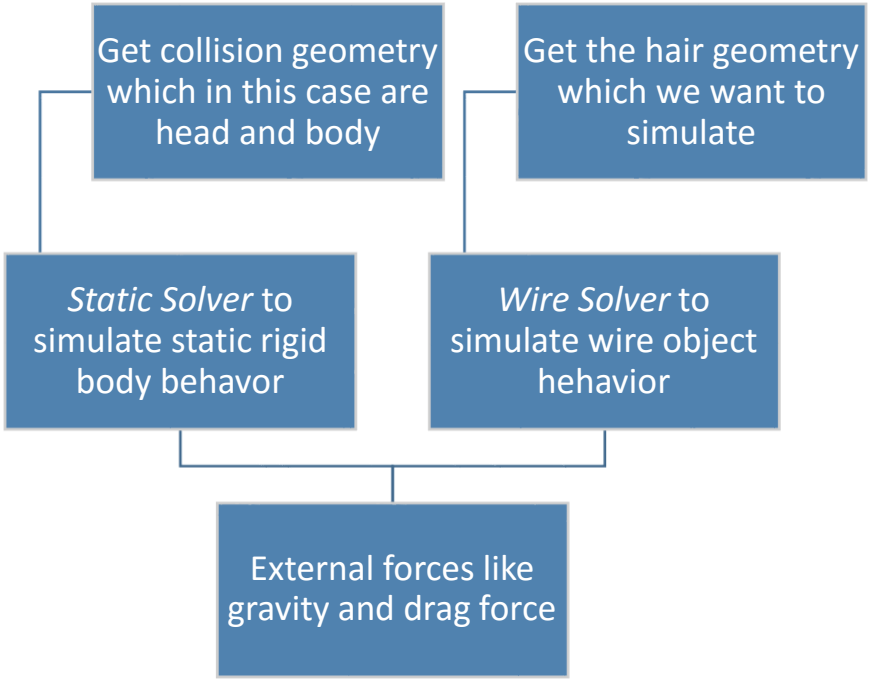


Figure 8.9: Essential steps inside the hair simulation DOP network.

Figure 8.10 is a screen shot of the hair constraint geometry. The constraints bind the root points to the head geometry to ensure they all follow the head rotation and translation.

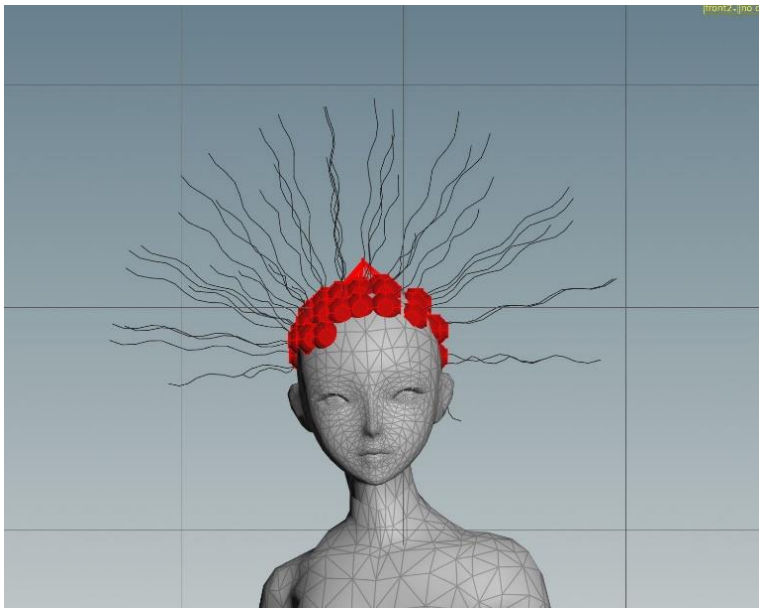


Figure 8.10: Constraints for attaching hair to head geometry.

8.2 Control field from hair simulation

The simulated hair geometries will be used to get the velocity control field for driving the fire simulation.

To get this velocity field, we resampled each hair curve to obtain 1000 points. This number can be changed by the user. We then apply a velocity attribute to these points and move them into a cylindrical space around the hair strand based on the method discussed in section 7.2. Artists have controls to customize this velocity field including adding noise and changing scale of the velocity at the different position along the hair curve.

The node network for obtaining the control field from hair geometry is shown in Figure 8.11. The detailed explanation of the function of each node is as follows:

1. *Resample* SOP. This node is used to resample the imported hair curve to obtain 1000 points. The hair curve will be divided equally into 999 segments.

2. *Convert* SOP. This surface operator converts the input data into the format the next step can use. For example, this node can convert the input from Houdini volume to polygon, or convert polygon into mesh etc. In our setup we set the output type as polygon. The input in our implementation is already polygon and this node will keep its format. The reason I include this node is for the sake of the robustness of the system. This will ensure proper functionality of the measure process in the next step since the *measure SOP* only work with polygons.

3. *Measure* SOP. The Measure surface operator can measure volume, area and distance and store the results as attributes. In this project I measured the direct distance from the first point to the last point of the hair curve and used this value in the vopsop in step 9.

4. *Attribute create* SOP. This surface operator is used for creating one or more attributes. In this project we created an attribute called *totalpoints*, the value of which is the total number of the points we wobtained from *resample* SOP, in this case 1000.

5. *Attribute promote* SOP. Inside Houdini attributes are stored in four different types: points, vertices, primitives, and details. The distance we got in step 3 is stored as a primitives attribute. However, the vopsop we are going to use in step 9 can only iterate on points attribute type. Because of this we need to convert the distance we get from step 3 from a primitives attribute to a points attribute.

6. *Reverse* SOP. The *reverse* surface operator can reverse the order of points. This node gives the user option to switch the direction of the velocity between root-to-tip or tip-to-root easily.

7. *Polyframe* SOP. This surface operator gets the tangent vectors of each hair curve point and uses these value as velocity vectors.

8. *Add* SOP. We use this surface operator to delete all the hair geometry but the points. We only need the points to get the velocity control field in the following process.

9. *Vopsop*. This vopsop contains a network of VEX operators to scatter the points into a cylindrical space and obtain new velocity values. The mathematics behind this network are explained in Section 7.2.

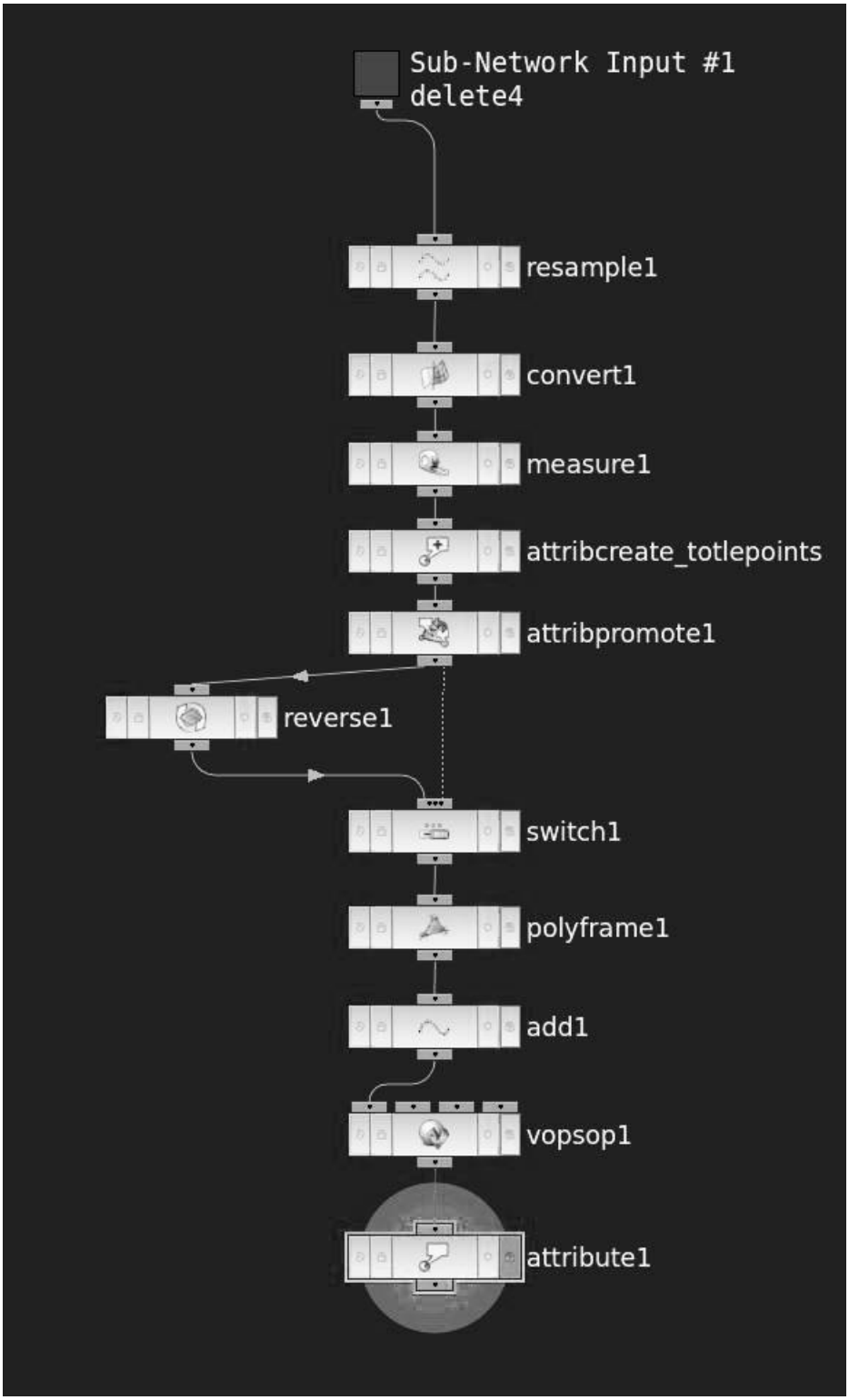


Figure 8.11: SOP network for obtaining a velocity field from hair curves.

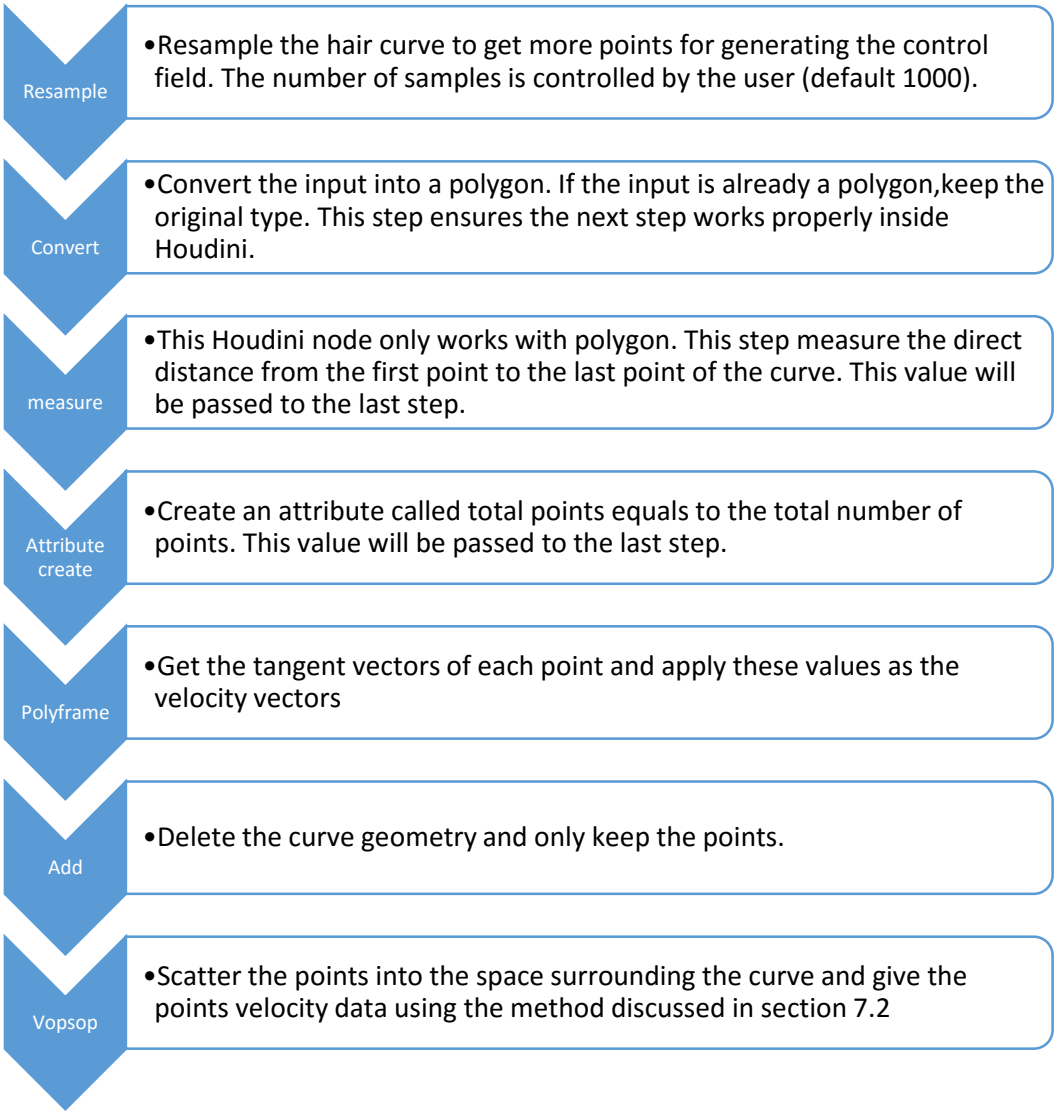


Figure 8.12: Flow graph for getting the velocity field.

Figure 8.13 shows what happens inside the vopsop. This VOP network is the Houdini implementation of the method discussed in section 7.2. The nodes inside the red area give each point a random displacement direction. Nodes inside the top blue area create a profile ramp to control the radius of the field, and the nodes inside the bottom blue area scale the velocity. Both of these ramp parameters are exposed to the users as

show in Figure 8.16, as *Tube shape* and *Velocity ramp*. The nodes inside the green area give the points randomized displacement distance.

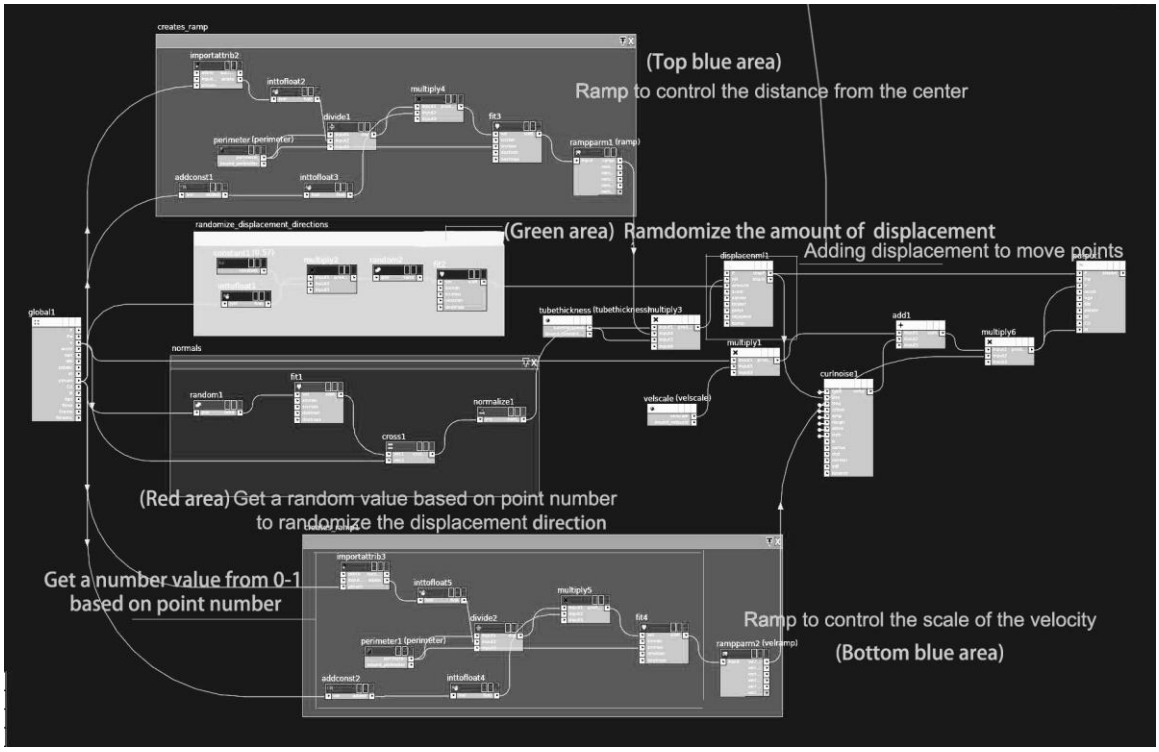


Figure 8.13: Network inside the vopsop. There are four modules highlighted in three different colors. The top blue area allows us to control the point's distance from its original position by creating a distance controlling ramp. The green area gives a random displacement distance. The red area generates a randomized direction for each point's displacement. The bottom blue area creates a ramp to control the scale of each point's velocity.

The Figure 8.14 shows the point cloud we get after scattering points into the cylindrical space around hair curves.

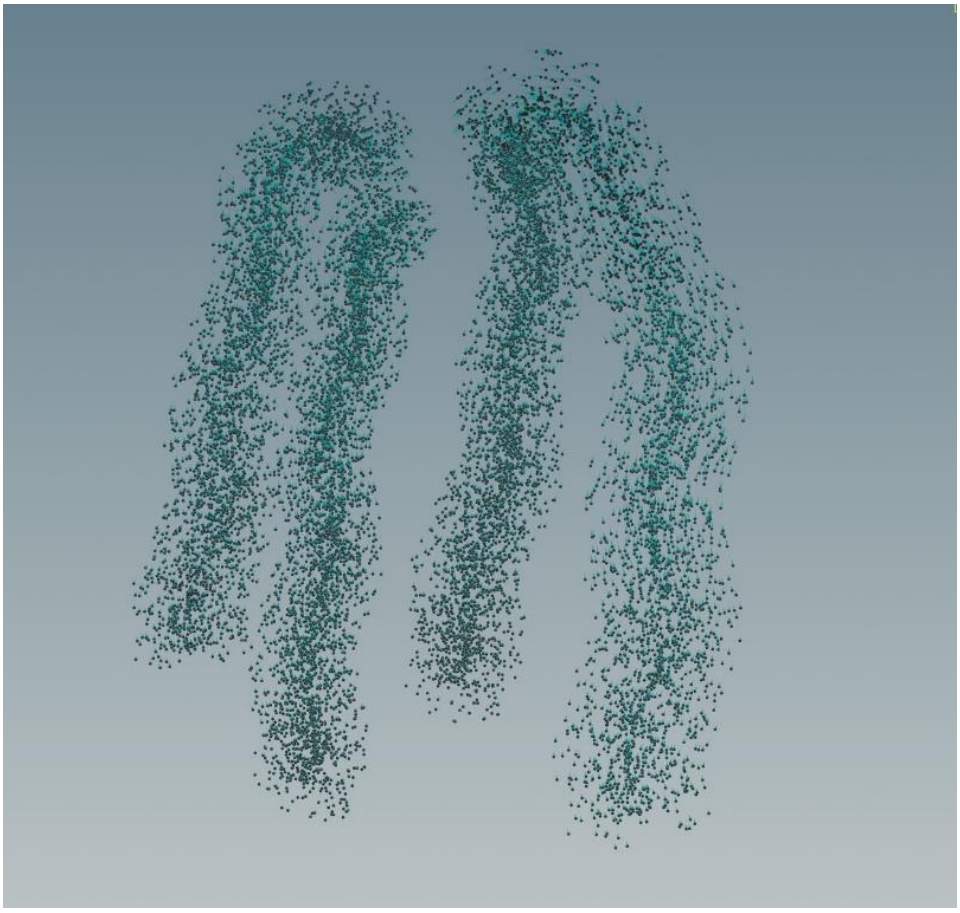


Figure 8.14: Point cloud with velocity attributes obtained from hair curves.

Each point has a velocity attribute. To show this attribute inside Houdini, we can use streamers to represent the direction and scale of the velocity. In Figure 8.15, the yellow color represent larger scale and red color represent smaller scale.

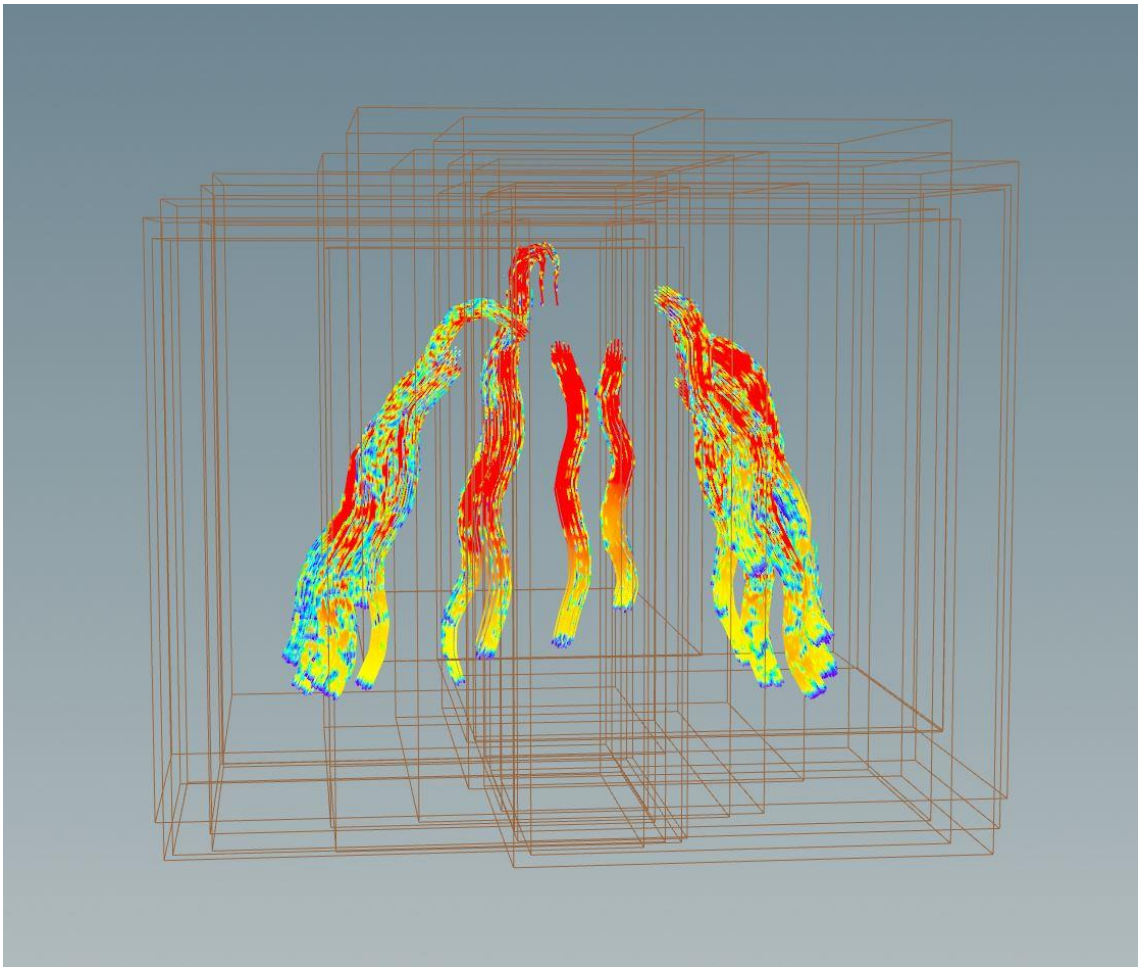


Figure 8.15: Velocity field obtained from vopsop. Red(darker) and yellow(brighter) curves indicate direction and scale of velocity. Yellow means the velocity has a larger value, red indicated smaller value.

The controls of the velocity field are exposed to the user as shown in Figure 8.16:

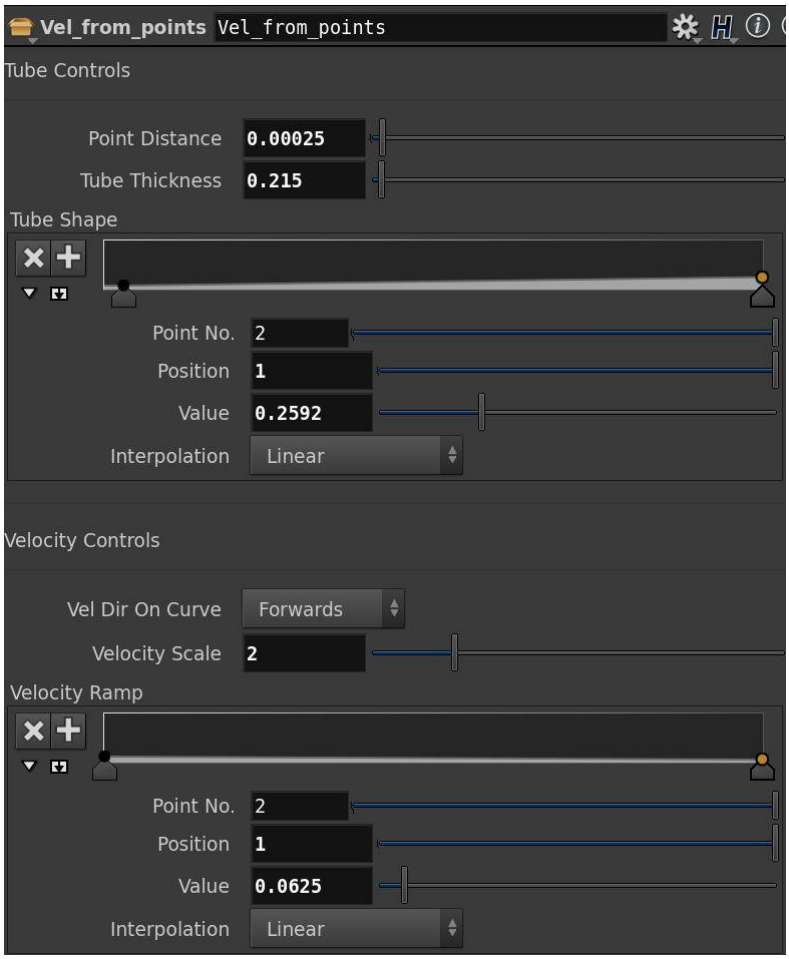


Figure 8.16: User control in creating velocity field.

Point Distance: controls density of the points cloud.

Tube thickness: it is a multiplier used to adjust the radius of the points cloud surrounding the hair curve. A larger value results in a thicker point cloud.

Tube shape: a profile ramp which controls the profile shape of the point cloud as well as the control field we create. Users can modify the value of the control to change the profile shape, user can also add control points to the ramp modifying the profile shape.

Interpolation: The interpolation menu gives users three different curves interpolation options. Default interpolation is linear.

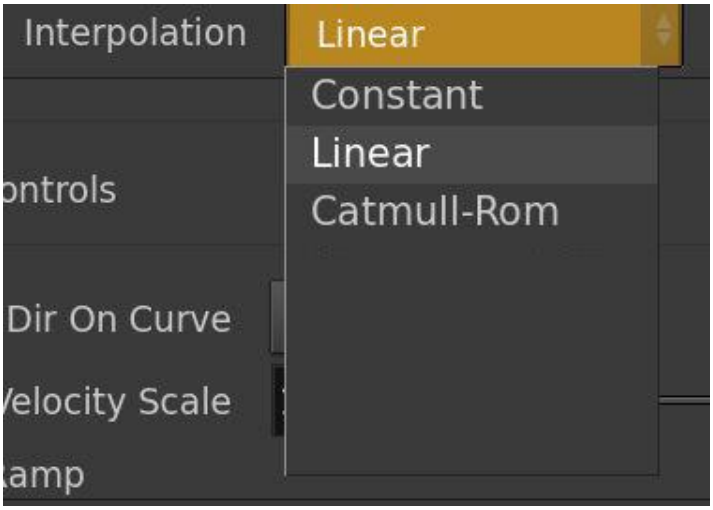
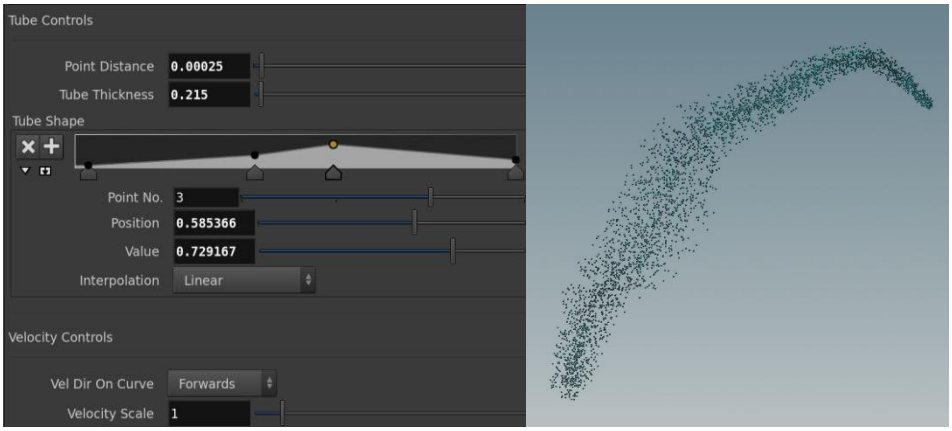


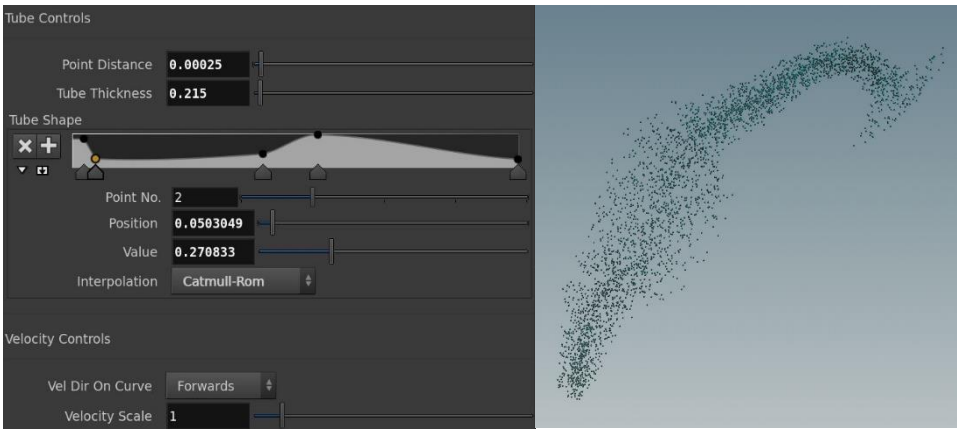
Figure 8.17: Interpolation drop list.

- *Constant* will have no interpolation-the profile shape is formed from several values steps.
- *Linear* will simply connect the points with straight lines.
- *Catmull-Rom* will use catmull-rom spline interpolation.

Figure 8.18 shows two different interpolations and the results generated.



(a)



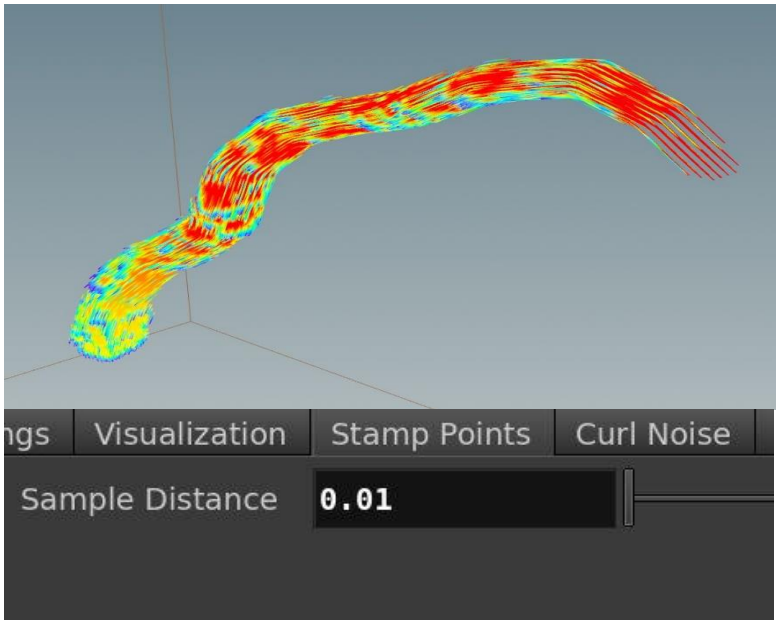
(b)

Figure 8.18: Different interpolation types and the point clouds generated.

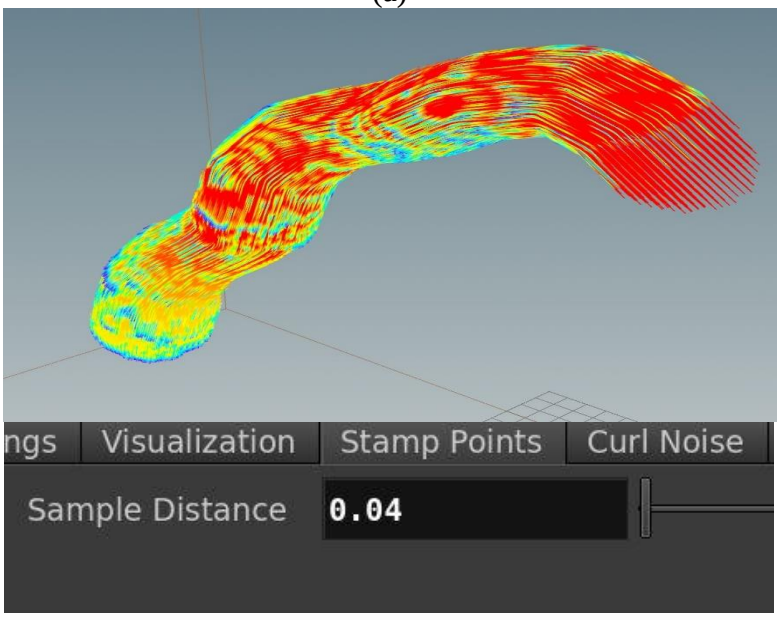
After obtaining the point cloud, we used a *fluid source* SOP (surface operator) to convert the velocity data of the point cloud into a *vel* field. Houdini uses this inside the DOPS (dynamics operator network for simulation) to represent velocity.

As shown in Figure 8.19, under the stamp points tab of the *fluid source* node we find a *sample distance* parameter. We use this parameter to further adjust the general thickness of the *vel* field. When more hair curves are added, the user should set the

sample distance parameter to a relatively small value to reduce the intersection of *vel* fields from different hair curves.



(a)



(b)

Figure 8.19: Change the *vel* field size using the *Sample Distance* parameter.

8.3 Fire simulation implementation

Instead of using the whole hair curve as a single fuel source, we replaced the hair curves with a number of small cylinders and used those as the fuel source of fire. We found that this method produced a more visually appealing result. This method allowed us to add volume noise to small independent sections of the source rather than giving a single volume noise to the whole strand. The reasons for this have been discussed in Chapter 7.

Figure 8.20: Screen shot of geometry used for creating fire source.

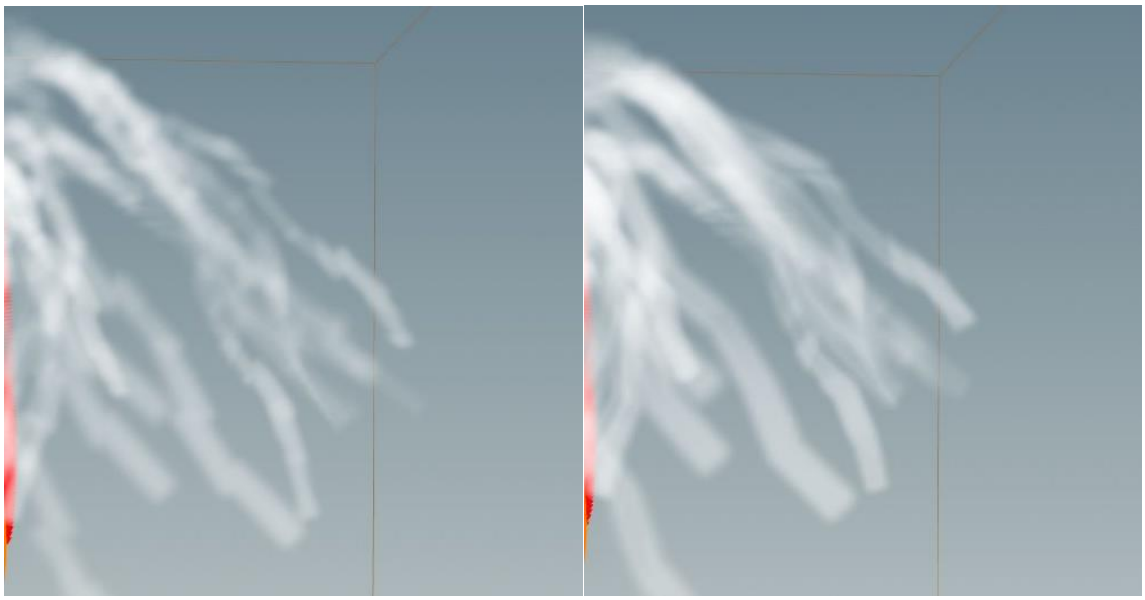


Figure 8.21: Fire source volume created. Image on the left is volume created using small cylinders; image on the right is volume created using the whole hair strand.

As shown by Figure 8.21, the left-hand side result shows the fire source with animated noise as applied to strand sections. The result on the right is the fire source with an animated noise applied to the whole hair geometry. We see clear and uniform edges in the right image, which will result in unnatural artifacts in our final fire renders. The Chapter 7 approach allows us to apply different noise to different segments. This helps in achieving more interesting fire simulation results.

The fire fuel source is now ready to be sent into the Houdini DOP (dynamic operator network) for fire simulation by using a *source volume* node. The vel field is also brought into the DOP network using the *source volume* node.

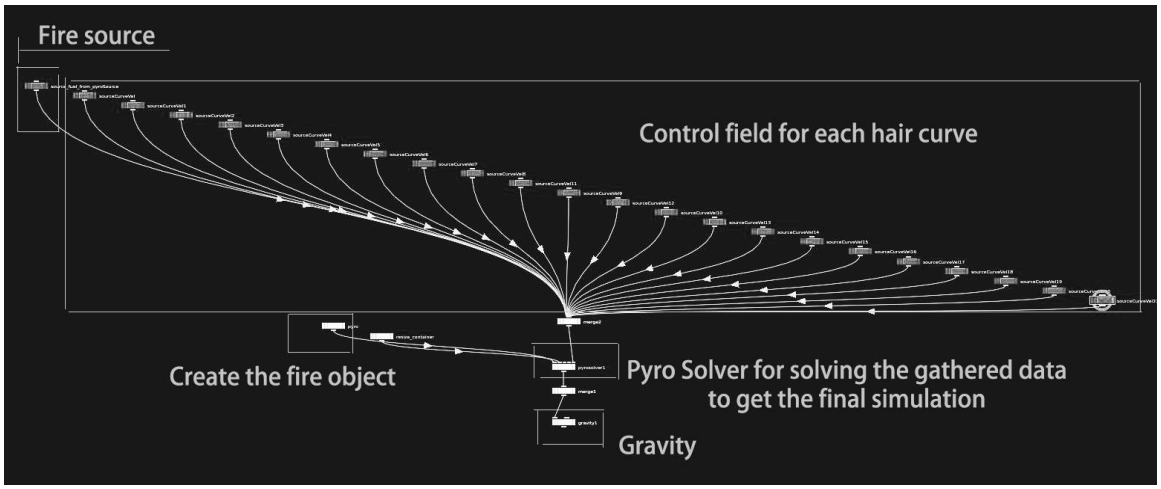


Figure 8.22: Screen shot of the node tree inside the DOP network.

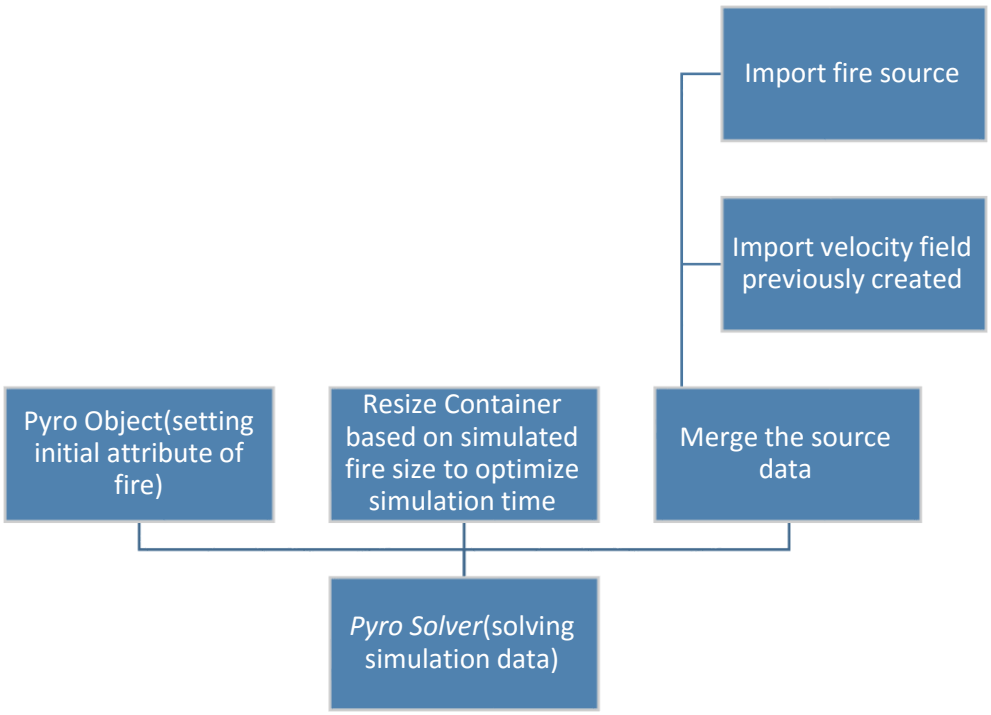


Figure 8.23 Processes inside the fire solver DOP network.

Figure 8.24 is a screenshot of the control panel of the fire source DOP (shown as the node in top left corner of Figure 8.22). We set the *initialize* mode to Source Fuel, meaning the dop network will bring in the hair-based density field to use as fuel for combustion. We set the parameter *Source Volume*, *Temperature*, and *Velocity* to "add". This will import related data from the fire source volume we previously created and add those attributes to our initial fire source. For the control field *vel* we just created, we set *Source Volume* and *Temperature* to "none" and set *Velocity* to "add". This is because the velocity is the only factor from *vel* field we want to affect our simulation. We merge all these source volume nodes together to be the source for our fire simulation.

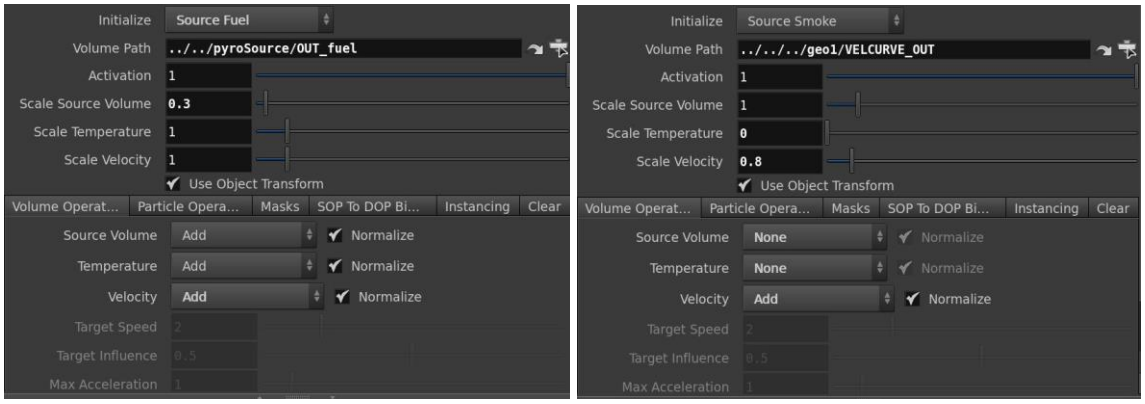


Figure 8.24: Different settings for source volume nodes. (a) shows the setting when used as the source of fuel. (b) shows the setting when used as the source of velocity field.

Using Houdini *Pyro Solver*, there are several controls for adjusting the shape of the simulated fire. Our fire-hair system doesn't expose all these parameters to the user, but we tested and set these parameters to get a best result. Here are some comparisons for each parameter alongside different settings:

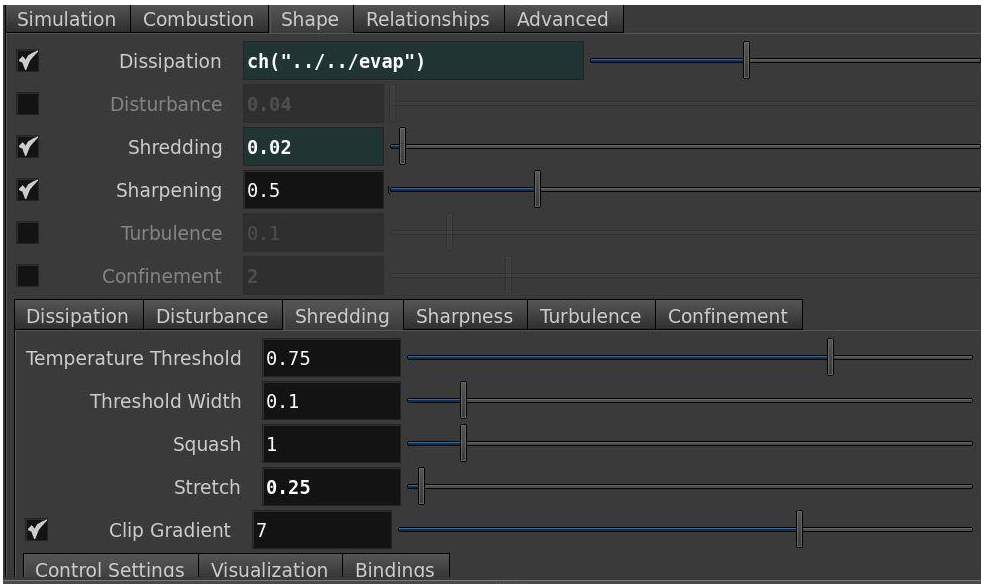


Figure 8.25: Setting for shape tab on *Pyro Solver*.

8.3.1 Dissipation

Dissipation is the parameter used to control how fast fluid disappears. In our project it will affect the length of the "fire" hair. Figure 8.26 shows comparison of differing dissipation settings.

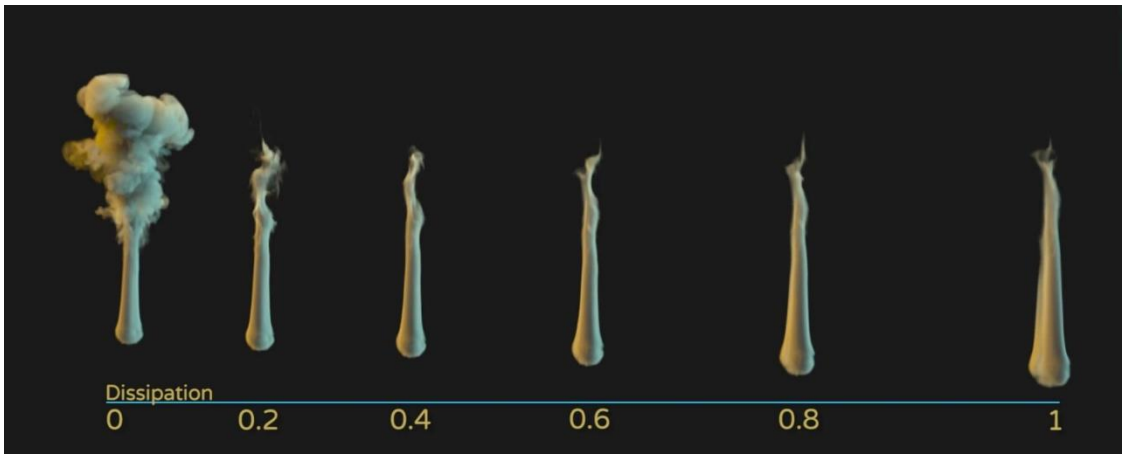


Figure 8.26: Simulation with different *dissipation* setting.

When the value is set to 0 the fluid never disappears. If the value is too high we run the risk of all interesting details disappearing due to the quick dissipation. After comparison we determined that 0.4 is a good value in balancing size and detail.

8.3.2 Shredding

Shredding pushes or pulls the velocity field based on the gradient of the heat field to create streaks, separation and fire "licks".

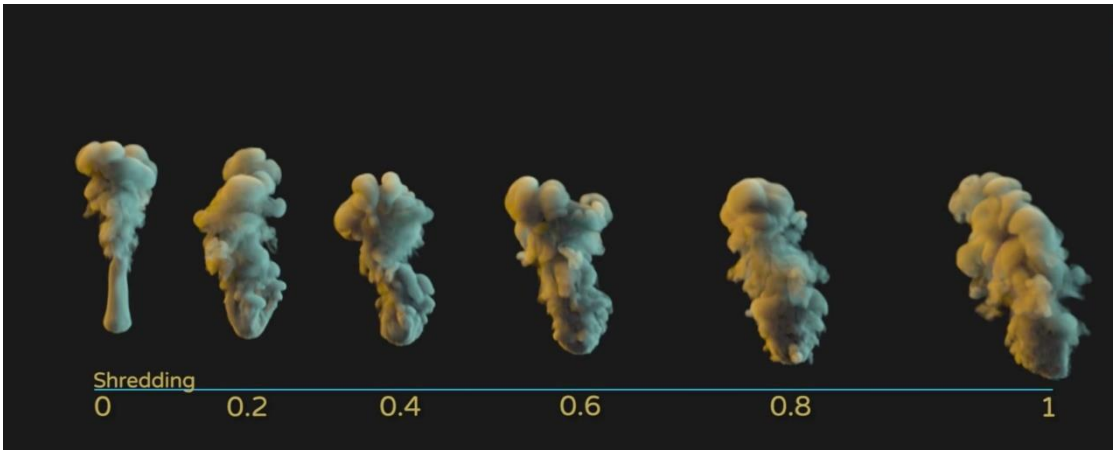


Figure 8.27: Simulation with different *shredding* settings.

Based on the scale of our simulation, I set the default *shredding* value to 0.02.

8.3.3 Sharpening

This *sharpening* features in the velocity field, producing wispier and streakier smoke rather than soft and fluffy smoke. This can be useful when using low shredding values to increase the definition of the features in the flame. However, over sharpening can give artifacts.

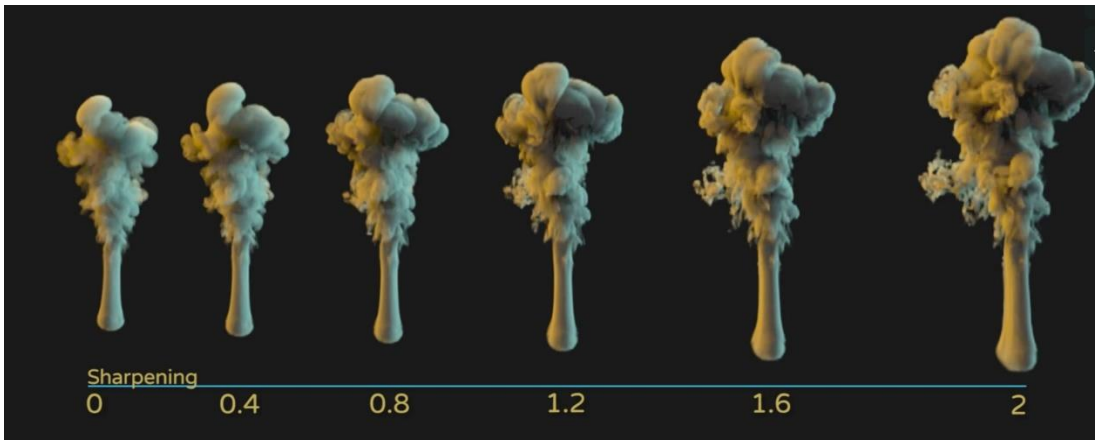


Figure 8.28: Simulation with different *sharpening* setting.

8.3.4 Disturbance

Disturbance influences detail within the smoke or fire without changing the simulation's general motion or shape.

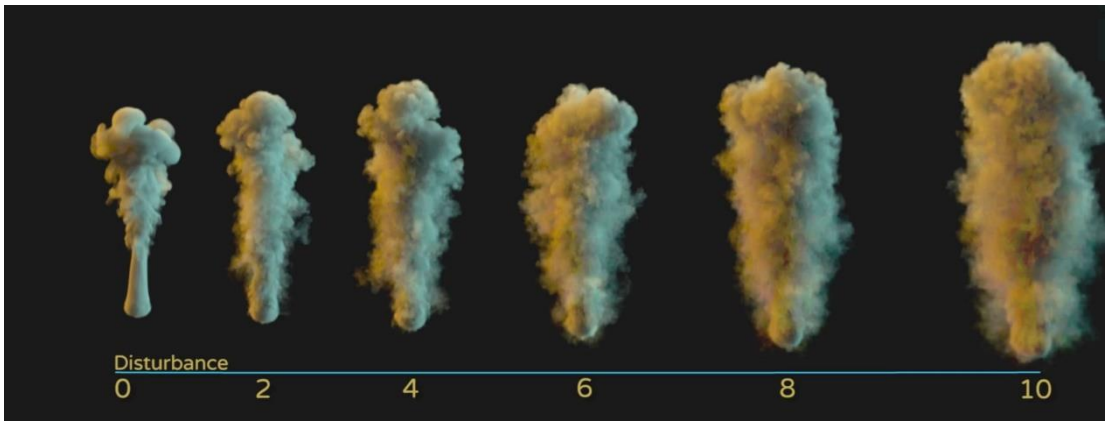


Figure 8.29: Simulation with different *disturbance* setting.

As one can see, setting the value to 0.4 give us a little bit of frizziness.

8.4 Render

We use three data fields from the simulation: heat, temperature and density. We deleted all other fields to limit needed data storage space.

We then used the *Flames* shader which is part of Houdini 14. We rendered the density field as smoke, the temperature field as the density of flame. The heat field controls the color ramp. All settings are shown in Figure 8.30.

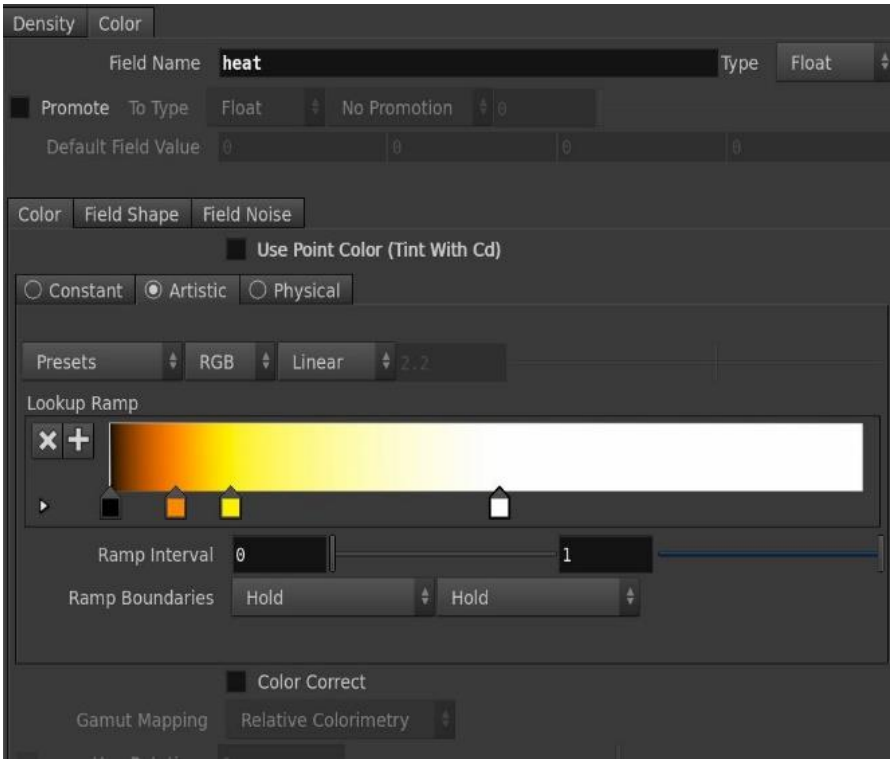
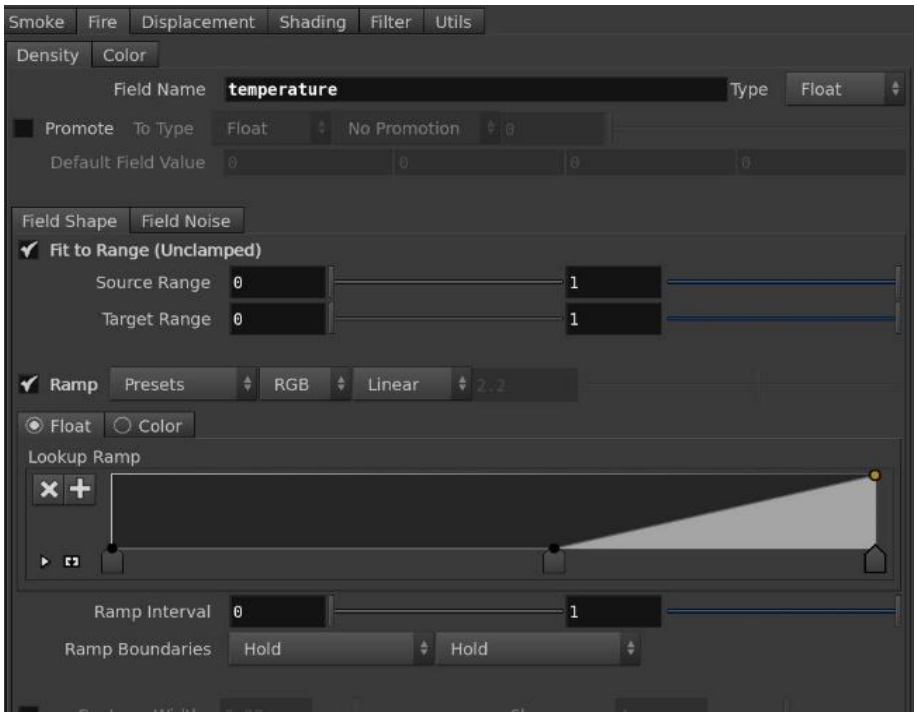


Figure 8.30: Shader setting.



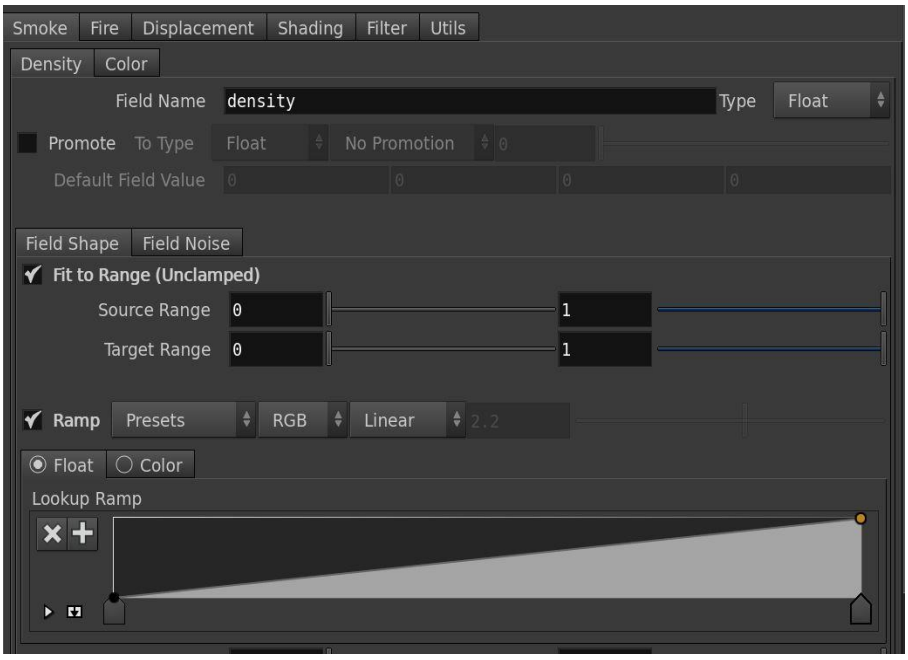
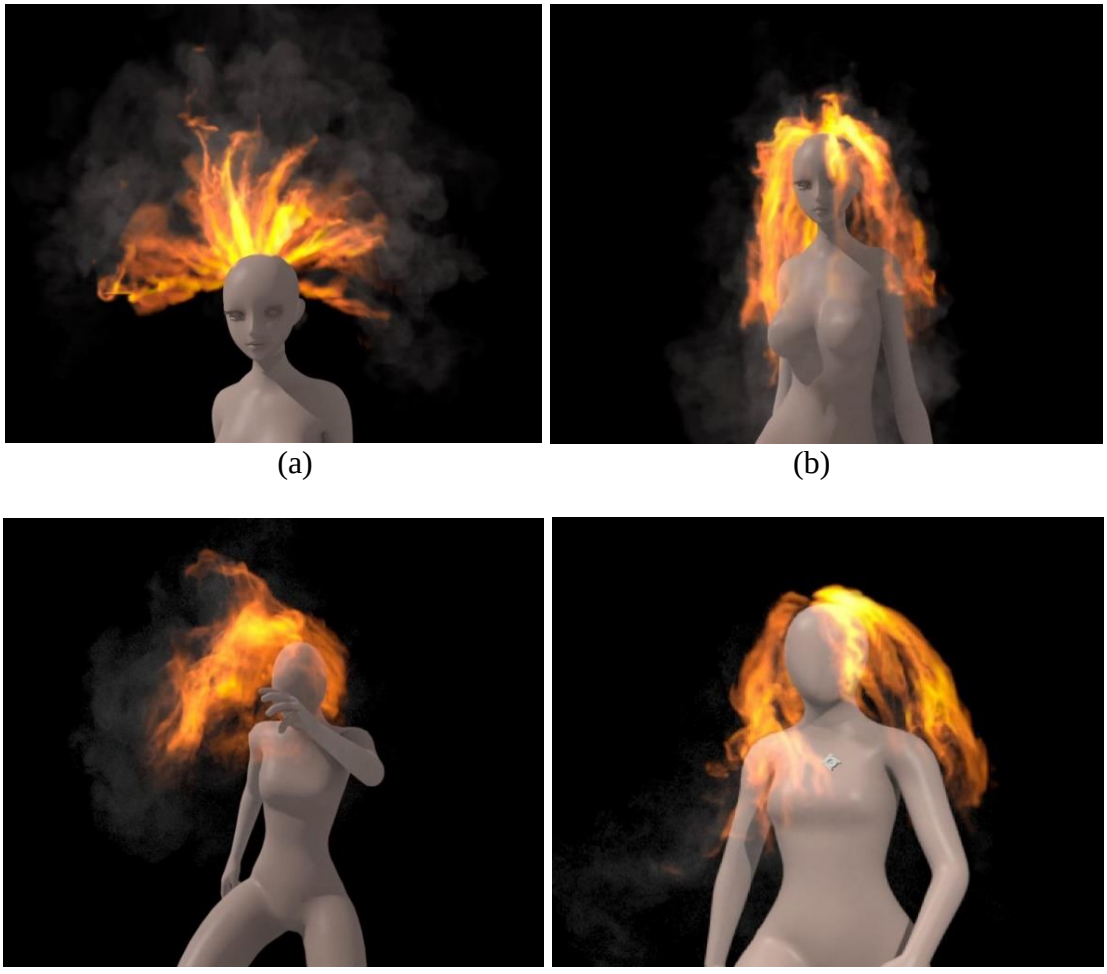


Figure 8.30 Continued.

8.5 Results

Our simulation was tested on two types of animation, *catwalk*, which has slow motion, and *samba dance*, which has intense and fast motion. For each animation, we also had two different types of hair, one is long and softer, another is short and stiffer.

The Figure 8.31(a) shows stiff mid-length hair with little curliness on a character with walking animation. The simulated hair curves mostly remain in their original position and the fire moves along the hair curves. We can clearly recognize the hair strands from the simulated fire. The Figure 8.31(b) shows soft long hair with a lot of curliness on the walking character. Hair strands are not as clear as the fire-hair in 8.31(a) but more desired curliness are shown in this final render. In Figure 8.31(c) we applied long soft hair to a dancing character. The fire is more dynamic due to the fast dancing animation. But the fire still represents the hair shape. Figure 8.30(d) shows long soft hair with little curliness on the character with walking animation. We see clearer hair strands compare to 8.31(b) while having more natural hair details than 8.31(a).



(a) (b)

(c) (d)

Figure 8.31: Final renders.

9. CONCLUSION AND FUTURE WORK

The most valuable achievement of this project was creating a reusable tool within Houdini, which artists can use to create art-directed fire-hair effects to apply to CG characters. The workflow is flexible and straight forward. Artists can customize the hair simulation by defining the length, shape and, stiffness of hair, and then generating the "fire hair" from the simulated hair.

We used digital paintings from the artists *sakimichan* and *Aster-phire* as visual references. Careful visual analysis was performed on the following aspects: real-life fire motion and color; static fire shapes from digital paintings and hair styles from digital paintings. This visual analysis abstracted the essential characteristics of the "fire hair" from the visual references. A hair simulation system was then developed to simulate basic hair motion. Then this hair simulation was used to control the fire simulation velocity control fields, thus giving us a fire simulation which is visually representative of the hair's shape and motion.

To obtain this velocity control field, we built a network, using vop operators in the SOP level, since vop compiles and executes much faster than a network using prebuilt nodes. This speeds up the simulation and shortening calculation time. The basic processes inside the vopsop are scattering the sample points and computing their velocity attributes.

After obtaining the point velocities we then use a *volume source* node to convert the points cloud into a volume field that can be used inside the DOP network to drive the fire simulation.

By combining the hair and fire simulations together we can create visually stylized “fire hair” effects. This system works effectively on both fast animation like *samba dance* and slow animation like *cat walk*.

This system helps artist avoid the Houdini operators network, which can be confusing. Thus it is easy enough for new learners to use. However, it can be a bit slow in generating the velocity field. And, the user still needs to export the simulated hair geometries into the fire simulation module.

9.1 Future work

Using the velocity field to control the motion of the fire has drawbacks. The desired control needs a stronger velocity field, leading to faster fire motion along the hair curves. Future research could focus on exploring alternative control methods to drive the hair simulation which would control the fire motion direction without affecting the fire motion speed.

The purpose of this project was to create a user-friendly and artistically-directed “fire hair” effects. Simulation is not necessarily the only approach. Further research can also aim to build a non-simulation method to achieve an even more controllable result. For example, this could take the form of animating noise added to the deformation of geometry, then procedurally giving it heat and temperature attributes. This would then

be rendered using a fire shader. This approach might give a faster workflow, avoiding the fire simulation.

Other future work could include using volumetric, rather than strand-based hair simulation, to create the fire control field.

REFERENCES

[1] J. Cameron, "The Abyss," Twentieth Century Fox Film Corporation, 1989.

[2] M. S. Johnson, "Ghost Rider," Columbia Pictures Corporation, 2007.

[3] Sakimichan, “Fire Vampire,” 2012.

[4] Aster-phire. "Flame Princess," http://aster-phire.deviantart.com/art/Flame-Princess-289390723.

[5] R. E. Rosenblum, W. E. Carlson, and E. Tripp, “Simulating the structure and dynamics of human hair: Modelling, rendering and animation,” *The Journal of Visualization and Computer Animation,* vol. 2, no. 4, pp. 141-148, 1991.

[6] Ken-ichi. Anjyo, Y. Usami, and T. Kurihara, “A simple method for extracting the natural beauty of hair,” in Proceedings of the 19th annual conference on Computer graphics and interactive techniques, 1992, pp. 111-120.

[7] “Modeling Dynamic Hair as a Continuum,” *Computer graphics forum,* vol. 20, no. 3, pp. 329-338, 2001.

[8] J. T. Chang, J. Jin, and Y. Yu, “A practical model for hair mutual interactions,” in Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation, San Antonio, Texas, 2002, pp. 73-80.

[9] F. Bertails, B. Audoly, Marie-Paule. Cani, B. Querleux, Frederic, r. Leroy, Jean-Luc. Leveque, “Super-helices for predicting the dynamics of natural hair,” *ACM Trans. Graph.,* vol. 25, no. 3, pp. 1180-1187, 2006.

[10] A. Derouet-Jourdan, F. Bertails-Descoubes, G. Daviet, Joelle Thollot, “Inverse dynamic hair modeling with frictional contact,” *ACM Trans. Graph.*, vol. 32, no. 6, pp. 1-10, 2013.

[11] "Side Effect Software," http://www.sidefx.com/.

[12] A. Ghourab, “A Fluid Implicit Particle Approach to a Pyro Solver in Houdini,” National Center for Computer Animation, Bournemouth University, 2011.

[13] R. Bridson, *Fluid Simulation for Computer Graphics*, p.^pp. 4, 2008.

[14] Mya Yee Win, “The Implementation of 2D Fluid Solver plugin for Houdini8.0,” National Centre for Computer Animation, Bournemouth University, 2007.

[15] M. Harris, “Fast fluid dynamics simulation on the GPU,” in ACM SIGGRAPH 2005 Courses, Los Angeles, California, 2005, pp. 220.

[16] R. Fedkiw, J. Stam, and H. W. Jensen, “Visual simulation of smoke,” in Proceedings of the 28th annual conference on Computer graphics and interactive techniques, 2001, pp. 15-22.

[17] N. Foster, and D. Metaxas, “Modeling the motion of a hot, turbulent gas,” in Proceedings of the 24th annual conference on Computer graphics and interactive techniques, 1997, pp. 181-188.

[18] B. E. Feldman, J. F. O'Brien, and O. Arikan, “Animating suspended particle explosions,” *ACM Trans. Graph.*, vol. 22, no. 3, pp. 708-715, 2003.

[19] M. D. Carmo. "What Causes Curly Hair?," http://www.yalescientific.org/2014/07/qa-what-causes-curly-hair/.

[20] R. Bridson, J. Houriham, and M. Nordenstam, “Curl-noise for procedural fluid flow,” *ACM Trans. Graph.*, vol. 26, no. 3, pp. 46, 2007.

[21] K. Perlin, “Improving noise,” *ACM Trans. Graph.*, vol. 21, no. 3, pp. 681-682, 2002.